UNIVERSIDADE DE LISBOA

FACULDADE DE CIÊNCIAS

DEPARTAMENTO DE INFORMÁTICA

STUDYING ELEMENTS OF GENETIC PROGRAMMING FOR MULTICLASS CLASSIFICATION

João Eduardo Silva Pombinho Batista

MESTRADO EM ENGENHARIA INFORMÁTICA
Especialização em Interação e Conhecimento

Dissertação orientada por:
Prof. Dra. Sara Guilherme Oliveira da Silva

2018

Are you doing anything next Saturday?

We could go for a walk, or watch ducks and talk.

Agradecimentos

First of all, I would like to thank my advisor Sara Silva for all her support through these last two years. I would also like to thank her for allowing me to freely explore any approaches I came up with and using a part of her time to discuss them with me. I would also like to thank her for, when I was having doubts, reminding me which path I wanted to take my life to, and supporting my decisions.

A special thank to all the faculty members who, directly or indirectly, helped me through these five years with words of support, suggestions, by making an example of how good professionals work, or by explaining to me how the academic world works. Namely, my advisor Sara Silva, Francisco Martins, João Marques Silva, Paulo Urbano, and António Branco.

I would also like to thanks everyone that both support and allowed me to delay the work done in my thesis. Namely, from the group *"Questões da Vida"*, Filipe Pereira, João Cardoso, João Antunes, João Rodrigues, and Dharmite Prabhudas, from the group *"Fculianos"*, João Becho, João Pinto, Tiago Correira, Francisco Araújo, Pedro Vieira, and Nuno Rodrigues, and everyone that I met during this last year, namely, Margarida Penedos, José Sousa, João Nobre, Kelly Silveira, Daniela Oliveira, and João Ye.

I would like to thank the Department of Informatics for allowing me to have the role of Monitor over this year and all my students, for their company over this year. Without them this year would have been way more monotonous.

A great thanks to those who took their time to read my thesis and give me suggestions on how I should improve my writing. Namely, my advisor Sara Silva, Filipe Pereira, Tiago Costa, Mário Carvalho, and Nuno Rodrigues.

Last but not least, I would like to thank those who, although I haven't been meeting very frequently, were inspiring, supporting, and always provided a great company. Namely, Luís Sampaio, David Silva, and my best friend, Ana Beatriz Alves.

Resumo

A classificação é tanto uma das mais fundamentais tarefas no que toca à tomada de decisões como um dos principais tipos de problemas que a aprendizagem automática tenta resolver. Ela encontra-se dividida em duas categorias, classificação binária e classificação multiclasse. O primeiro tipo pode ser facilmente resolvido de várias formas utilizando métodos já disponíveis. Por outro lado, a classificação multiclasse requer métodos mais especializados, devido à alta complexidade dos problemas desta natureza. Outro problema que este tipo de classificação enfrenta é que métodos como redes neuronais ou florestas aleatórias, apesar de serem os métodos que actualmente fornecem melhores resultados, podem não oferecer modelos facilmente interpretáveis.

Na implementação padrão de Programação Genética (PG), descrita em *Genetic Programming: vol. 1, On the programming of computers by means of natural selection* (1992) [3], cada indivíduo é representado como uma árvore em que os nós não terminais contêm funções e os nós terminais contêm valores numéricos ou índices para as características de cada amostra, permitindo-lhes calcular valores em $\mathbb{R}$ quando lhes são dadas amostras de um conjunto de dados. Estes indivíduos conseguem facilmente resolver problemas de regressão linear e de classificação binária. A classificação binária é tipicamente resolvida de uma forma simples. Por exemplo, num problema com uma classe A e uma classe B, o individuo pode associar a amostra à classe A se o valor obtido for negativo, caso contrario associará a amostra à classe B. Em (J. R. Koza, 2010) [7], é possível ver que a PG tem ótimos resultados nestes dois tipos de problemas. No entanto, como foi dito anteriormente, um problema de classificação com múltiplas classes requer métodos mais especializados para obter bons resultados, fazendo com que a implementação padrão de PG não seja adequada para este tipo de problemas.

Apesar da PG nunca ter sido o método mais adequado para resolver problemas de classificação multiclasse, em (V. Ingalalli, 2014) [8], foi proposto um novo método, baseado em PG, que não só consegue ter bons resultados em classificação multiclasse, consegue também devolver modelos interpretáveis. Este método, chamado M2GP (Multidimentional Multiclass Genetic Programming), é uma variante do algoritmo tradicional em que cada indivíduo em vez de ter apenas um nó na sua raiz, tem vários nós. Esta alteração fez com que os indivíduos em vez de devolverem um valor em $\mathbb{R}$, devolvessem um valor em $\mathbb{R}^n$. Agora os indivíduos conseguem ter um espaço de saída com n dimensões onde

são representadas as amostras do conjunto de treino. A geometria deste espaço de saída permite que seja feita uma abordagem de classificação baseada em agregados, onde cada classe é representada pelo conjunto de coordenadas obtidas pelo indivíduo quando lhe são dadas as amostras dessa classe. Neste tipo de classificação cada ponto é associado à classe cujo centroide se encontra mais próximo. O M2GP consegue ter resultados comparáveis aos dos perceptrões multicamadas e das florestas aleatórias. Desde a sua criação foram feitos melhoramentos ao algoritmo como o M3GP [4], o eM3GP [2] e o M4GP [20].

Neste projeto, iremos estudar o algoritmo M3GP. Este algoritmo é semelhante ao seu antecessor M2GP. A diferença entre as duas abordagens encontra-se no número de nós que um indivíduo tem quando é criado e como é que esse número varia ao longo das gerações. No M2GP, os indivíduos começam todos com um número preestabelecido de nós e esse número é mantido até ao final do treino. No M3GP, cada indivíduo começa com apenas um nó na sua raiz, tal como no algoritmo padrão de PG, de forma a procurar soluções em espaços com menos dimensões nas primeiras gerações. À medida que as gerações passam, são usados operadores genéticos que permitem adicionar e remover dimensões aos indivíduos, permitindo-lhes explorar espaços em dimensões superiores.

Apesar do M3GP dar resultados melhores que o seu antecessor, este continua a poder ser melhorado. A implementação original deste algoritmo avalia os indivíduos da população com base no número de amostras corretamente classificadas no conjunto de treino. Este método de avaliação tem dois problemas. O primeiro é que uma avaliação deste tipo não incentiva que haja uma evolução suave do espaço geométrico gerado por este indivíduo. O segundo problema deve-se à complexidade da função utilizada para associar amostras a uma classe. Este algoritmo utiliza a distância de Mahalanobis [12] para associar as amostras à classe do centroide mais próximo. Usando esta distância, a complexidade do cálculo da distância de um ponto a um centroide em $\mathbb{R}^n$ é de $\mathcal{O}(n^3)$, enquanto que a complexidade usando a distância Euclidiana é de $\mathcal{O}(n)$. O método usado para selecionar os operadores genéticos também pode ser melhorado. Na implementação original são usadas probabilidades fixas para escolher os operadores genéticos. Apesar desta abordagem dar bons resultados, da mesma forma que existem operadores que são inúteis na fase inicial da evolução, pode haver operadores que só são úteis em fases iniciais da evolução. Um exemplo seria o operador que remove nós da raiz dos indivíduos no algoritmo M3GP, se os indivíduos começam com apenas uma dimensão, este método é inútil na primeira geração. Selecionar estes métodos em etapas de evolução em que eles têm uma probabilidade reduzida de aumentar os resultados da avaliação dos indivíduos pode, no mínimo, atrasar a evolução da população.

A primeira etapa deste projeto foi a implementação e validação do M3GP. Esta foi feita em Java [40] e tentou seguir à risca todas as especificações do algoritmo M3GP, descritas em *M3GP - Multiclass Classification with GP* (2015) [4], de modo a que esta pudesse ser validada replicando os resultados exibidos no artigo. Apesar dos indivíduos

obtidos na nossa implementação serem maiores e com um maior número de dimensões, seguimos para a segunda e terceira etapas deste projeto quando os indivíduos mostraram resultados que não eram necessáriamente piores em termos de número de amostras corretamente classificadas, tanto no conjunto de treino como no conjunto de teste. A diferença no tamanho dos indivíduos pode ser devido ao M3GP original ter sido implementado usando o GPLAB [39], que utiliza medidas adicionais de controlo de inchaço [37, 38] por defeito.

A segunda etapa deste projeto foi a criação de novas funções de avaliação. Estas funções foram desenhadas com o objetivo de adotar uma função de avaliação que permitisse que a evolução do espaço de saída dos indivíduos fosse mais suave. Para tal, em vez de usar uma função de avaliação baseada no número de amostras do conjunto de treino corretamente classificadas, passamos a usar uma função baseada em distância que tenta afastar os centroides dos agregados de amostras enquanto tenta puxar as amostras para o seu respetivo centroide. Havendo a necessidade de obter resultados em tempo útil, em vez de distância de Mahalanobis, foi usada a distância Euclidiana. Os resultados obtidos foram comparados com uma função semelhante à usada no algoritmo original, com a diferença de que esta usa uma função que associa as amostras ao centroide mais próximo usando a distância Euclidiana.

Das duas funções testadas nesta etapa, uma delas deu resultados péssimos e foi imediatamente ignorada. A outra função testada deu resultados equivalentes à função usada como ponto de comparação, não só no número de amostras corretamente classificadas, como no tamanho dos indivíduos, no número de dimensões, e na evolução em geral, enquanto tem uma complexidade computacional inferior à função base. Durante esta fase tivemos um problema com a criação da função de avaliação dos indivíduos. As distâncias tendem a ser maiores em espaços geométricos com um número de dimensões mais elevado, como tal foi necessário pensar numa forma de normalizar distâncias entre indivíduos com diferentes numeros de dimensões. Esta tarefa foi feita com sucesso mas acreditamos que existirão funções que consigam fornecer melhores resultados.

A terceira etapa deste projeto foi alterar o método de seleção dos operadores genéticos de modo a que o programa inicialmente desse uma igual probabilidade a cada operador genético de ser selecionado e, com o passar das gerações aprendesse que operadores genéticos melhoram, ou não, os indivíduos. Desta forma, os operadores que estiverem a ser benéficos para os indivíduos da população terão uma maior probabilidade de serem selecionados, enquanto que os operadores prejudiciais irão ter uma probabilidade reduzida. Numa segunda sub-etapa, foram também criados novos operadores genéticos, alegadamente maus, com o objetivo de estudar a evolução das suas probabilidades.

Os resultados obtidos além de indicarem melhorias significativas no número de amostras corretamente classificadas em metade dos conjuntos de dados utilizados, também indicaram que o cruzamento da versão padrão de PG é sempre útil e a sua utilidade tende

a aumentar com o passar das gerações. Também indicam que os operadores que trocam dimensões entre individuos tendem a perder a sua utilidade com o passar das gerações. Uma conclusão tirada nesta etapa final que serve de confirmação para observações feitas nas outras etapas é que as populações tendem a usar mais vezes o operador genético que muta o indivíduo adicionando uma dimensão quando o conjunto de dados tem muitas classes. Por fim, foi proposto um novo método de cruzamento que troca as dimensões entre três indivíduos. Este método, por ter uma maior probabilidade de seleção em todos os conjuntos de dados, mostrou ser preferível ao cruzamento que troca as dimensões entre dois individuos.

Palavras-chave: Programação Genética, Aprendizagem Automática, Classificação, Multi-classe, Aglomeração Multi-dimensional

Abstract

Although Genetic Programming (GP) has been very successful in both symbolic regression and binary classification by solving many difficult problems from various domains, it requires improvements in multiclass classification, which due to the high complexity of this kind of problems, requires specialized classifiers.

In this project, we explored a multiclass classification GP-based algorithm, the M3GP [4]. The individuals in standard GP only have one node at their root. This means that their output space is in $\mathbb{R}$. Unlike standard GP, M3GP allows each individual to have n nodes at its root. This variation changes the output space to $\mathbb{R}^n$, allowing them to construct clusters of samples and use a cluster-based classification.

Although M3GP is capable of creating interpretable models while having competitive results with state-of-the-art classifiers, such as Random Forests and Neural Networks, it has downsides. The focus of this project is to improve the algorithm by exploring two components, the fitness function, and the genetic operators' selection method.

The original fitness function was accuracy-based. Since using this kind of functions does not allow a smooth evolution of the output space, we tried to improve the algorithm by exploring two distance-based fitness functions as an attempt to separate the clusters while bringing the samples closer to their respective centroids.

Until now, the genetic operators in M3GP were selected with a fixed probability. Since some operators have a better effect on the fitness at different stages of the evolution, the fixed probabilities allow operators to be selected at the wrong stages of the evolution, slowing down the learning process. In this project, we try to evolve the probability the genetic operators have of being chosen over the generations. On a later stage, we proposed a new crossover genetic operator that uses three individuals for the M3GP algorithm. The results obtained show significantly better results in the training set in half the datasets, while improving the test accuracy in two datasets.

Keywords: Genetic Programming, Machine Learning, Classification, Multiclass, Multidimensional clustering

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The focus of this project is the study and improvement of the multiclass classification algorithm M3GP (Multidimentional Multiclass Genetic Programming with multidimensional populations), proposed in *M3GP - Multiclass Classification with GP* (2015) [4].

This chapter will first give a brief introduction about what Genetic Programming (GP) is, and then, we will explain why it's worth to do research on Multiclass classification, why we are using genetic programming to do this kind of classification, how we will improve the M3GP algorithm, and what changes this project will bring to the GP field and the machine learning community.

1.1 Genetic Programming

As stated by Koza in *Genetic Programming: On the programming of computers by means of natural selection* (1992) [3], *"In nature, biological structures that are more successful in grappling with their environment survive and reproduce at a higher rate. Biologists interpret the structures they observe in nature as the consequence of Darwinian natural selection operating in an environment over a period of time. In other words, in nature, structure is the consequence of fitness. Fitness causes, over a period of time, the creation of structure via natural selection and the creative effects of sexual recombination (genetic crossover) and mutation. That is, fitness begets structure."*. In nature, each individual has their own structure and behavior, even within the same population. Since each individual need to execute several tasks in order to survive and reproduce, individuals who are better prepared to execute these tasks will have a higher chance of survival and reproduction while the other individuals, who are less fit for survival, will have a lower chance of survival and reproduction. This is the concept of survival of the fittest and natural selection described by Charles Darwin in *On The Origin of Species by Means of Natural Selection* (1859) [32]. Over many generations, the population as a whole comes to contain more individuals whose structure and behavior enable them to better execute their tasks and to survive and reproduce. Over time, the structure of the individuals changes because

of this natural selection that tries to raise their fitness.

Having this in mind, Genetic Programming (GP) is a machine learning approach where the main algorithm follows the principles of the Darwinian natural selection. This algorithm first receives a task and then creates a population of randomly generated programs that will try to solve it. Although the most likely scenario is one where all these individuals will have a bad result in solving their task, some of these individuals will achieve better results and therefore a better fitness. The selection of individuals for reproduction favors the individuals with higher fitness. This way, those individuals will be more likely to reproduce, leaving their offspring to the next generation. After a new generation of individuals is completed, these individuals are evaluated, and the process starts all over again, trying to improve the results of the population over the course of the generations.

1.2 Motivation

Classification is not only one of the most fundamental tasks in order to make decisions in real-world problems, it's also one of the main type of problems in which machine learning is used. This creates a great interest in improving the available methods to solve this kind of problems. Classification can be divided into two categories, binary classification, and multiclass classification. The first type can be solved in endless ways using many different available methods. Multiclass classification requires more specialized methods due to the high complexity of this kind of problems. Another problem it faces is that the best methods used, normally Neural Networks and Random Forests, don't generate models that are ready to be easily interpreted by humans.

Although GP has not always been an adequate method for multiclass classification, in (V. Ingalalli, 2014) [8], a new GP-based method was implemented. This method not only gave good results in multiclass classification, it also provided interpretable models. This algorithm is a variation of the standard GP in which an individual is able to convert each sample into a point in $\mathbb{R}^n$. The geometric properties of this kind of output space allow the individuals to use a cluster-based classification. This classification is made by creating a cluster for each class using the samples on the training set, converted to coordinates in the output space, an then associating the samples to the closest class centroid using the Mahalanobis distance[12]. This method, named M2GP (Multidimentional Multiclass Genetic Programming), had results comparable to those of the Multilayer Perceptron and the Random Forests. Since then, there have been a few variants the algorithm such as the M3GP [4], the eM3GP [2] and the M4GP [20].

The M3GP algorithm has two issues that motivate the study of an alternative fitness function. In its original state, M3GP uses a fitness function that not only doesn't take into consideration the intra-cluster and inter-cluster distances but is also very expensive

in terms of computational complexity. This function uses the Mahalanobis distance which is very complex. To calculate the distance between two points in $\mathbb{R}^n$, this function has a complexity of $\mathcal{O}(n^3)$, while the Euclidean distance has a complexity of $\mathcal{O}(n)$.

Another issue with GP algorithms is that since the population selects the crossover and mutation genetic operators with a fixed probability, they might be selected on stages of the evolution where they are not needed. Using these genetic operators on the wrong stages can slow down the evolution. The variance of the importance of each genetic operator over the generations was already an issue in standard GP. Since the M3GP algorithm has more genetic operators, this problem is bound to be worse in this algorithm. This motivates us to try to solve this issue by adapting the probabilities that each genetic operator has of being selected to the needs of the population.

1.3 Goals

This project had three goals: 1) to implement the M3GP algorithm and replicate the results reported in (L. Muñoz, 2015) [4], 2) to make improvements in the fitness function of the M3GP algorithm, and 3) to automatically adapt the probabilities that each genetic operator has of being used during the evolution. These goals will be mentioned later as, respectively, the first, second, and third stages of this project.

The fitness function used by M3GP in (L. Muñoz, 2015) [4] was the classification accuracy. The accuracy of an individual was obtained by classifying each sample of the training set by associating it with the closest class centroid using the Mahalanobis distance. In order to make an improvement to the fitness function, we are making an attempt to use a distance-based fitness function rather than an accuracy-based fitness function. To obtain results quickly the fitness functions we are testing use the Euclidean distance. The results are compared with the results obtained using an accuracy-based fitness function that uses the Euclidean distance.

To automatically adapt the probabilities that each genetic operator has of being used, we are changing the method of selection of the genetic operator to one that initially gives each genetic operator an equal chance of being selected. From this point, every time an individual uses an operator which improves the individual's fitness, the probability of occurrence of that genetic operator will increase, otherwise, it will decrease. If the operator doesn't create changes on the individual's fitness the probability will also decrease to avoid choosing useless operators. With this, the operators that are beneficial to the evolution will be used more often, while the prejudicial operators will be avoided, hopefully speeding learning.

1.4 Contributions

This project offers the following contributions:

- A full implementation of the M3GP (M2GP with Multidimensional Populations) method, written in Java [40] and ready to be used on Weka [6].
- New, and more efficient in terms of computational complexity, fitness functions were implemented and tested on the M3GP algorithm. A study over the results obtained by testing these fitness function was then made, in order to understand the differences in the accuracy, and on the evolution of the population when using different fitness functions.
- New methods for the automatic adaptation of the probabilities of selection of genetic operators were proposed and tested. A study was made on the influence of these adaptations on the accuracy of the population and on the evolution of the probabilities each genetic operator has over the generations.

In this project, we made a contribution to the GP community by validating the previously obtained results using the M3GP, and by presenting alternatives that increase this algorithm's efficiency by using different fitness functions and methods to select genetic operators. Since there are not many robust methods for multiclass classification that produce an easily interpretable model, the improvement of the M3GP algorithm is an important contribution to the machine learning community in general.

1.5 Structure of the document

This document is organized with following structure:

- Chapter 2 - Describes some applications of the M3GP algorithm and other related implementations in real life problems. Some other methods with different approaches are also mentioned such as non-clustering based GP methods and methods that rely on genetic programming to do feature evolution but use other clustering methods for classification.
- Chapter 3 - Contains information about the datasets and parameters used in each simulation and information about the algorithm used. The algorithm section is divided into three parts, the original algorithm, the modifications made to test new fitness functions, and the automatic adaptation of the genetic operator probabilities.
- Chapter 4 - Describes the implementation made to run the classifier. It also includes a resume with the purpose of each of the classes used.

- Chapter 5 - Contains the results of all the runs made in this project and their analysis.
- Chapter 6 - Contains the conclusions of the project and discusses possible future work.

The later sections of this document are the bibliography, glossary, and the appendices.

Chapter 2

Related work

In this section, we will mention some applications of GP algorithms such as classification, feature evolution, and evolution of the probabilities of selection of genetic operators. The topics related with classification discussed in this section are the M3GP algorithms considered to be similar, other GP-based algorithms that use clustering techniques, non-clustering GP algorithms for multiclass classification, and real-world applications of GP algorithms to solve classification problems. There are many articles referring ways to use GP in multiclass classification, and many of these approaches use GP for feature evolution, so the explanations about these topics will be brief. A longer explanation will be given on the papers related to the evolution of operator probabilities, as the topic will be studied further in this project and it seems to be a topic of importance with few recent works.

2.1 M3GP and other variants

Since there are many variants of the M3GP classifier, this section will cover some of those variants, namely its predecessor M2GP, its successor M4GP, its brother eM3GP, and another approach which can be considered as similar to the M3GP classifier for being cluster-based classifiers and using a similar method of classification.

A new GP framework, named Multi-dimensional Multiclass Genetic Programming (M2GP), was proposed in (V. Ingalalli, 2014) [8]. At the time, this method was a novel algorithm for tree-based GP where each individual could have n nodes at its root instead of only one, allowing the individuals to have an output space in $\mathbb{R}^n$. The geometry of the output space allowed this GP method to use a cluster-based classification method and solve multiclass classification problems. This approach was tested on a large set of benchmarks problems from several different sources and was able to compete against the Multilayer Perceptron and the Random Forests, showing GP was able to solve this kind of problems.

The M3GP algorithm is very similar to the M2GP algorithm, the main difference

being in the evolution of the number of dimensions in each individual. While the M2GP uses a fixed number of dimensions, established in the initialization of the classifier, the M3GP creates all individuals with only one dimension and the mutation operators allow each individual to create or destroy dimensions.

M4GP was presented in (La Cava W., 2017) [20] as a new classification method to evolve feature transformations. This is a stack-based GP system that allows each individual to produce multiple outputs. Like M2GP and M3GP, this method has the advantage over typical classification methods of producing models that are easily interpretable by humans. The results obtained indicate that M4GP outperforms other GP methods for classification and performs competitively with other machine learning methods in terms of accuracy.

Like many other classifiers, M3GP appears to be suffering from overfitting, and is negatively affected by class imbalance, and also suffers from bloat on a dimensional level. A new method named ensemble M3GP (eM3GP) was proposed in (S. Silva, 2016) [2], intended to address some of the M3GP issues. Some variants of the eM3GP were tested and the results observed had competitive results with classifiers like Random Forests, Random Subspaces, and Multilayer Perceptron.

In (Smart W., 2004) [16], the authors describe a probability-based GP approach to multiclass classification problems. Although it's not explicit that clustering methods are being used, the approach has several similarities with M2GP. This approach, instead of directly associating a sample to the class of the closest class centroid on the output space, uses a Gaussian distribution to create probabilities of the samples being in a point in space, and associates each sample to the class it has the higher probability of belonging to. The results suggested that this approach is more effective and more efficient than the basic GP approach.

2.2 Other Genetic Programming clustering methods

Not all cluster-based methods are as simple, or direct, as the one used in M2GP, or in M3GP. In this section, some other methods that use different cluster-based approaches are mentioned, namely a graph-based clustering method and a lattice-based clustering method.

In (Andrew L., 2017) [17], the authors propose a method that uses GP for graph-based clustering (GPGC). This approach performs a graph-based clustering that does not require that the number of clusters is set in advance. GPGC was compared with a number of well-known methods on a large number of datasets that vary on the number of samples, features, and classes. The results indicated that GPSC is the most generalizable of the methods that were tested, achieving good performance across all datasets. It's worth mentioning that GPSC outperformed all the tested methods on the hardest ellipsoidal

datasets, without needing the user to pre-define the number of clusters. To the knowledge of the article's authors, this was the first work which proposed using GP for a graph-based clustering.

In (C. Wu, 2013) [18], the authors discuss the performance of regression-based coordinate transformations for GPS applications. Then, for building better regression models for coordinate transformation, they develop and integrate with GP a lattice-based clustering method. This method partitions the GPS application into lattices that will be used as clusters that have their sizes determined by the geographic locations and distributions of the GPS reference points. Each cluster of lattices serves as a training dataset for the genetic regression model of coordinate transformations. This way, the data points contained in each lattice can be considered to be of the same importance. With this, the biased regression results, caused by the imbalance distribution of data, can also be eliminated. The experimental results show that the proposed method can further improve positioning accuracy than previous methods.

2.3 Non-clustering GP for multiclass classification

Not all methods for multiclass classification rely on clustering techniques. In this section, some other methods such as linear GP, an approach based on dividing multiclass classification problems into binary classification problems, and an approach based on a voting scheme, will be mentioned.

An alternative form of GP, Linear GP (LGP), is studied in (Downey C., 2010) [22]. LGP demonstrates a great promise as a classifier since the division of classes is inherent in this technique. Two new crossover genetic operators that significantly improve the performance of the classifier were developed in this article by combining biological inspiration with detailed knowledge of program structure. The first genetic operator mimics biological crossover between alleles, which helps reduce the disruptive effect on building blocks of information. The second genetic operator is an extension of the first where a heuristic is used to predict offspring fitness guiding search to promising solutions. The results obtained indicate that the novel crossover operators improve the performance of the LGP algorithm on the tested datasets.

A common approach for binary classification is to use a threshold of the output value of an individual. For example, if the classifier uses a function that outputs a number when it receives a sample from a dataset, we can classify the sample as one class when this value is negative and as the other when the value is not negative. Although this approach is commonly used for binary classification, it is not practical for multiclass classification. The method proposed in (Smart W., 2005) [24] uses a variation of this approach, used to solve binary classification problems, to solve multiclass classification problems. It divides a multiclass classification problem into many binary classification problems. This

classifier, Communal Binary Decomposition (CBD), was compared with two other GP-based classifiers, Program Classification Map (PCM) and Probabilistic Multiclass (PM) showing good results in all datasets, having the best test accuracy in three of the four used datasets. A similar approach was also used in (K. Liu, 2009) [25].

In (Zhang M., 2009) [26], the authors discuss an approach where each individual produce an output for each class. These output values are meant to be used in a voting scheme to determine the class of the sample that was given to the individual. This approach was examined and compared with the standard GP approach in four multiclass classification problems with increasing difficulty. The results obtained indicate that, in these problems, this approach outperforms the standard GP approach. Another advantage this method has is that the program structure allows to easily produce multiple outputs for multiclass classification problems while keeping the advantages of the standard GP approach for an easy crossover and mutation.

2.4 Real world applications of GP in clustering techniques

In this section, we will mention a few real-world applications for clustering techniques. Although these applications are all related with GP, it is only used to tune the clusters into some more easily separable ones and is not used for classification. Other already well-known classifiers are then used after this tunning.

In (F. Ratle, 2008) [13], the authors study a GP-based approach with the intention of tuning the affinity matrix of a spectral clustering matrix. This was made using a database with clusters confirmed by a police investigation, used to assess the potential of the analysis of the chemical signature of heroin and cocaine in the investigation process. The results obtained were compared to standard methods that are used in the field of chemical drug profiling and indicate that conventional approaches miss the inherent structure in the data, which is highlighted by methods such as spectral clustering and its variants.

In (J. P. Papa, 2016) [14], the authors study unsupervised land-use / land-cover using K-Means. Although this method already plays an important role in the pattern recognition community, there is always room for improvement. One problem usually found is that the dataset samples are not near the centroid of the cluster, which may increase the difficulty a program has in learning a dataset. With this in mind, the approach in this paper was using a GP-based algorithm in order to enhance the K-Means effectiveness by minimizing the distance of the samples to the centroid of each cluster. This allowed a better separation of the cluster and consequently a better classification of the dataset samples.

In (N. P. Shetty, 2016) [15], the authors make an attempt to use GP with two objectives. The first was to remove unnecessary features from the samples of a dataset. The second objective was to convert the dataset into clusters, one for each different class the dataset has. The dataset used was the KDD Crup'99 [9]. This dataset contained samples with

features from different kinds of informatic intrusions or the lack of intrusion and a label for their respective kind of intrusion. This dataset contained some noise and unnecessary features. The clusters obtained by this process were then classified by both K-Means and K-Medoid. The results obtained indicate that K-Medoid has a significantly better accuracy than the K-Means in this given dataset.

It's worth to mention that not all applications of GP rely on other methods. In (A. Patnaik, 2016) [21], a GP-based clustering method was used to define the level of service (LOS) criteria of urban roads in India using multiclass classification. This classification was made using a dataset that had the physical properties of the streets and other features such as the average speed of the traffic and the pedestrian activity. Unlike the cases previously mentioned that rely on other methods to do the classification, this approach was based solely on GP. The results of this research were well received and suggested that the road network needed geometric improvements in order to produce a better quality of service.

2.5 Feature Evolution with Genetic Programming

Besides classification and symbolic regression, another application of GP is feature evolution. Feature evolution tries to create a new set of features from the initial one. This is accomplished by removing unnecessary features and/or by synthesizing new features. The objective is to create a new set of features that are easier for the program to learn, or simply to know if a feature is needed. As stated in [36], a reason that leads to a classifier success or failure is the quality of the features used. If these features correlate well with the output, the dataset can be easily learned. If the features do not correlate with the output in a simple way, the classifier can have some difficulties learning the dataset. Feature engineering helps the classifier by synthesizing valuable features from the original features, that might not have been useful while separated. It also solves the dimensionality problem by synthesizing new features from more than one of the original features, or simply by removing unnecessary features. This also helps the classifier by reducing the number of features that are used to learn the problem. Since a dataset with d binary features would need to be tested in 2^d samples, making it exponentially harder to verify as the number of features increases, this reduction of the number of features is highly beneficial to the classifier. In this section, some projects related to feature engineering will be mentioned.

The authors present in (Y. Zhang, 2009) [27] a generic feature extraction method for pattern classification using multi-objective GP. This method is able to evolve the near-optimal set of mappings from pattern space to a multi-dimensional decision space while optimizing the dimensionality of that decision space. This framework evolves feature extractors that maximize class separability. The efficiency of this approach was demon-

strated by making statistically-founded comparisons with a wide variety of established classifier paradigms over a range of datasets. The results showed that this method delivers statistically smaller misclassification errors. At the very worst, these methods displayed no statistical difference in a few comparisons.

The extraction of features for classification is often performed heuristically, despite the effect this step has on the performance of the classifier. The authors present the Evolutionary Pre-Processor in (J. Sherrah, 1998) [28]. This is an automatic non-parametric method for the extraction of non-linear features. This method uses GP to pre-process the data to improve the performance of a classifier. This method was tested on nine real-world datasets and was able to increase the test set accuracy when compared to the classification of the original samples. The dimensionality of the data used was decreased and the features not required for classification were removed. This Pre-Processor was also able to behave intelligently by deciding where it should perform feature extraction or perform feature selection.

Two GP-based approaches are proposed in (N. Al-Madi, 2013) [23]. The first approach, GP-K, uses the K-means clustering technique in order to transform the produced value of GP into class labels. The second approach, GP-D, uses a discretization technique to perform a transformation, equivalent to the one performed by GP-K. After a comparison made between GP, GP-K, GP-D, and other state-of-the-art classifiers, using both binary and multiclass datasets, the results showed good improvements in terms of accuracy when compared to the original GP. GP-D achieved higher accuracy values than those of the original GP as well as the GP-K. The comparison with the state-of-the-art classifiers revealed competitive accuracy values.

In (L. Guo, 2011) [29], the authors apply GP to perform automatic feature extraction from original feature database used in this experiment, with the aim of improving the discriminatory performance of a classifier while reducing the input feature dimensionality. In experiments on two common epileptic EEG detection problems, the classification accuracy of the KNN classifier on the GP-based features is significantly higher than on the original features. Simultaneously, the dimension of the input features for the classifier is much smaller than that of the original features.

2.6 Adaptation of Genetic Operator probabilities

A common problem in GP is knowing when each genetic operator is needed. Some genetic operators can have a more positive impact on the initial stage of the evolution and others on later stages. As we will see in this project, the effect each genetic operator has on each stage of the evolution may vary from dataset to dataset. The usual approach is to use a fixed probability to select each genetic operator. This section describes work related to adapting the probabilities of selecting each genetic operator to the need of the

population.

This subject has already been studied before in (A. Tuson, 1998) [30]. However, unlike what will be done in this project, the author used Genetic Algorithms (GA), not GP. Unlike GP, which is very robust in terms of parameters, as will be explained in this section, GAs are very sensitive to their parameters, needing to be adapted to each problem. In this dissertation, the author states that the automatic adaptation of probabilities could benefit the population in three criteria. By reducing the amount of time that is spent finding suitable values for GA control parameters, by increasing the quality of solutions obtained, and by allowing the GA to find a solution of a given quality more quickly. To study the effects varying the operators' probabilities have on the GA performance, five test problems were selected, each with differing properties. The conclusions taken were that the operator probabilities used have a significant effect upon GA performance, and appropriate values vary from problem to problem and that the population model used has a big impact both upon performance and upon the behavior of the GA with different operator probabilities.

Tuson's dissertation [30] also touched a similar subject, not the adaptation of genetic operator probabilities but the adaptation of the genetic operators. The conclusions taken from this study are that adapting the genetic operator is not necessarily a good thing. It's also mentioned that these adaptations should be made outside that main GA algorithm after analyzing the information on the datasets and that if improvements in performance occur, they are likely to be in the speed of search, which can result in a possible detriment of solution quality.

In (Niehaus J., 2001) [31], the authors make an attempt to reduce the number of free parameters within GP without reducing the quality of the results by adapting the operators' probabilities. This was a common procedure in areas such as Evolution Strategies, and Genetic Algorithms, but GP had very few attempts with this procedure. In this paper, three different methods of adaptation were tested.

The first method, Population-Level Dynamic Probabilities (PDP), states that the probability that each genetic operator has of being used, increases with the number of consecutive uses that increase the individual's fitness. This method uses a counter to know how many times a genetic operator has successfully increased the fitness of an individual. This counter is reset if the operator fails to increase the individual's fitness. The probability each genetic operator has of being selected is given by the following expression 2.1, where n is the number of genetic operators and p_i is the probability of the operator i.

$$p_i = p_{all} + \left\lfloor r_i \frac{100 - n \cdot p_{all}}{\text{scale}} \right\rfloor \tag{2.1}$$

where

$$p_{all} = \left\lfloor \frac{20}{n} \right\rfloor,\ r_i = \frac{\text{success}_i^2}{\text{used}_i},\ \text{scale} = \sum_{j=1}^{n} r_j\ .$$

The second method, Fitness Based Dynamic Probabilities (FBDP), was created after the initial experiments have shown that different operators have different success rates depending on the fitness of their parent individual. This method gives different probabilities to the genetic operators based on their previous success on the individuals with different fitnesses, having a different probability for individuals with low or high fitness. Since the article does not seem to be very explicit on how this probability evolve, the equations will not be mentioned.

In the third method, Individual-Level Dynamic Probabilities (IDP), each individual j has two arrays of values. One where its kept the genetic operator's probabilities and another, filled with counters cnt, that contain the number of consecutive uses of each genetic operator on the individual that did not improve its fitness. Each counter is reset if its respective GO is able to improve the individual's fitness. The probabilities each GO has of being used are calculated using a relation between their counter's value and the sum of all GO' counters on the individual. The probability p_i that each genetic operator has of being selected is given by the expression 2.2.

$$p_i = p_{all} + \left\lfloor \frac{(\max_{1\leq k\leq n} cnt_j^k + 1 - cnt_j^i)(100 - n \cdot p_{all})}{n(\max_{1\leq k\leq n} cnt_j^k + 1) - \sum_{k=1}^{n} cnd_j^k} \right\rfloor, \tag{2.2}$$

where

$$p_{all} = \left\lfloor \frac{20}{n} \right\rfloor .$$

These three methods were compared to the standard method of selection and the results suggested that the average fitness of the population was higher when an adaptative method was being used. Out of these three methods, the one with the best result was the IDP. The reasoning used by the author was that the PDP uses the same probability for all individuals and a good genetic operator for an individual with a good fitness may not be good to an individual with lower fitness. The FBDP solves this problem by showing different probabilities for individuals with different fitnesses but it fails at evolving larger individuals where their structure is different and the same operator in two individuals can have different results. With IDP, every individual has its own probabilities evolved, learning what is good or bad for its fitness.

When working with evolutionary computation (EC), it is necessary to tune many parameters within the algorithm. In (M. Sipper, 2018) [33], the authors discuss an approach for an automatic parameter-seeking process.

This method tried to optimize the values for the population size, number of generations, crossover rate, mutation rate, and tournament size, by using a meta-level genetic algorithm over these parameters. This meta-GA's population's individuals select these parameters from a predefined range of values and use them to evolve a population. The evolution of this GA uses the results of n runs as its fitness, other than that, is very similar to any other GA algorithm, not being worth to mention the evolutionary algorithm.

The results obtained from this experiment indicate, unlike the common approach of focusing on predefined ”good” values tends to suggest, there is a large range of good parameter sets. The conclusions related to the size of the population, and the number of generation, were that increasing these values does not necessarily lead to fitness improvements. The results should be good as long as they are not both low. The authors refer to (Arnold C., 2017) [34], mentioning that recent findings suggest the use of fewer generations. The conclusion related to the tournament size was that although the values usually used are from 3 to 7, using higher values also give good results. Lastly, the conclusions related to the crossover and mutation rate are that these values can have many diverse values, as long as they are not both small as that would decrease the variation of the population over the generations.

Chapter 3

Methodology

3.1 Datasets

The first stage of the project was the implementation of the M3GP algorithm and the replication of the results reported in *M3GP – Multiclass Classification with GP* (2015) [4]. In order to validate our implementation, we ran the algorithm with the same parameters and datasets as those used on the article. These datasets included a nice variability on the number of samples, features, and classes, and therefore they were maintained for the remainder stages of the project. They are HRT (Heart), SEG (Image Segmentation), VOW (Vowel), YST (Yeast), M-L (Movement-Libra) and WAV (Waveform) which are found in the UCI dataset repository [9] and the IM-3 and IM-10, from the U.S. geological survey (USGS) earth resources observation systems (EROS) data center (EDC) [10]. Their number of classes, attributes, and samples are refered in Table 3.1.

Data Set	HRT	IM-3	IM-10	M-L	SEG	VOW	WAV	YST
No. of classes	2	3	10	15	7	11	3	10
No. of attributes	13	6	6	90	19	13	40	8
No. of samples	270	322	6798	360	2310	990	5000	1484

Table 3.1: Datasets used and their number of classes, attributes, and samples

3.2 Parameters

Since this project started with the comparison of our implementation with the original M3GP work [4], the parameters used were also the same. They are listed in Table 3.2 and briefly described in this section. For a more comprehensive explanation of the M3GP algorithm, the reader is refered to the next section.

The parameters determining the composition of the individuals are the function set and the terminal set. The function set includes the sum, subtraction, multiplication and

protected division, in order to ensure the closure property [3]. The difference between the normal division and our protected division is that if the divisor is 0, this division returns the dividend. The terminal set contains the indices to the features of the data and random constants between 0 and 1.

At the start of each run, the data set is shuffled and the first 70% of the samples are selected as the training set. A population containing 500 individuals is then created using the Full method [3] with a depth of 6. After its creation, the population evolves until either reaching the 100^{th} generation or the best individual of a generation achieves a perfect accuracy on the training set.

During evolution, the best individual is selected to be pruned and the second best is selected by elitism to move to the next generation while the remainder of the new population is filled by selection individuals using a tournament of size 5 and applying one of two crossover genetic operators or one of the three mutation genetic operators to create a new individual. The individuals have a maximum depth of 17, which is enforced by discarding any individuals that have a depth above this value when they are created.

All these parameters were maintained over the project with the exception of the probability that each genetic operator (GO) has of being picked. In the third stage of this project, the boundary between mutation and crossover genetic operators is removed and each genetic operator will start with an equal probability of being selected. This probability is meant to evolve for each individual over the generations.

Function set	+, -, *, // (protected division)
Terminal set	Index of features $\cup$]0,1[
Training set size	70% of the data set
Population size	500
Population initialization method	Full
Initial maximum tree depth	6
Maximum tree depth	17
Number of generations	100
Tournament size	5
Elitism	1
Probability of crossover	0.50
Probability of mutation	0.50

Table 3.2: Parameters used on the runs by default

3.3 Algorithm

3.3.1 Elements of the M3GP algorithm

The initialization of our implementation of the M3GP is very similar to the one used in standard GP and specified in *Genetic Programming: On the programming of computers by means of natural selection* (1992) [3], and was made an attempt to be the same used in (L. Muñoz, 2015) [4]. In this sub-section, we will describe the elements of our approach when implementing the M3GP algorithm: we will describe how it is initialized, how the individuals are evaluated, how the individuals are selected and evolve, what is the function of the pruning GO, and how the elitism was implemented. In this sub-section, we may not give details about some parameters that were already mentioned in 3.2. We will mention some of the new genetic operators, but further details about them will only be given in 3.3.3. We should mention that the new mutation GOs are allegedly bad and were only implemented in order to test the behavior of M3GP when it's given the chance to avoid using these operators.

Initialization: In order to search for more simple solutions first, each individual is created with only one branch at its root. This means that in the first generation, all individuals have only one dimension. To promote diversity among the initial population, every individual is created using the Full method. In other words, the initial population would be the same in standard GP with the exception that the root node is a dummy node whose only purpose is to agregate the branches that represent each dimension. This dummy node is never changed by the genetic operators.

Evaluation: When an individual is created, it converts the training dataset into points in a $\mathbb{R}^n$ space, where the centroid and covariance matrix of each class are then calculated. Using these two elements, when a sample from a dataset is given to the individual, the Mahalanobis distance from the sample to each centroid is calculated, and the closest centroid is selected class as the predicted class. Another variation was made in this project. This variation used the Euclidean distance instead of the Mahalanobis distance. Other than this, the fitness function is equivalent. The individuals are evaluated using an accuracy-based fitness function. This kind of fitness function considers an individual superior to another if it has the highest accuracy. If they tie on accuracy, the smaller individual is preferred. The individuals could also be evaluated using a distance-based fitness function that tried to increase the distance between centroids while decreasing the distance that each sample has to their respective centroid. These fitness functions will be mentioned later in 3.3.2.

Selection for breeding: The parents of the next generation of individuals are selected by tournament. Each tournament receives the population and randomly selects five individuals. From these individuals, the one with the highest fitness is the winner of the tournament, and a future parent.

Crossover: This implementation uses two crossover methods. Both methods receive two individuals. The first method (**St-XO**) randomly selects a node in each individual and swaps the nodes, just like the subtree crossover method proposed by Koza for the standard GP. The second method (**Swap-dim**) randomly selects a dimension in each individual and swaps the dimensions. At the third stage of this project, a third crossover method (**Swap3-dim**) was implemented. This method randomly selects a dimension in each of three individuals and moves the first dimension to the second individual, the second dimension to the third individual and the third dimension to the first individual.

Mutation: This implementation contains three mutation methods. The first method (**St-Mut**) is the mutation used in standard GP. It randomly selects a node within the individual and replaces the node with a new tree generated using the Grow method. The second method (**Add-dim**) adds a new dimension to the individual. This dimension is a new tree generated using Grow. The third method (**Rem-dim**) removes a randomly selected dimension from an individual with at least two dimensions. If the individual has one dimension and the third method is selected, the individual is returned unmodified. At the third stage of this project, four other mutation operators were implemented. One removed all dimensions except one and turns that dimension's root node into a terminal node. Another randomly selects a dimension and turns its root node into a terminal node. The other two methods randomly select a node within an individual. One turns the node into a terminal node if it's not terminal and if the node is terminal it's replaced by a new tree, generated using Grow. The other operator turns the node into a terminal node. A more extensive description of these new mutation genetic operators is written in 3.3.3.

Pruning: Starting with the first dimension of the individual and ending with the last dimensions. This method evaluated the individual without one of its dimensions at a time. If its fitness improves with the removal of this dimensions, this dimensions is removed permanently. If the individuals have only one dimension, this method does nothing to the individual. Due to the high computational complexity of this operator, it is applied only to the best individual of each generation.

Elitism: Besides the pruned individual, the second best individual is always selected to move to the next generation without any modifications made.

Genetic Operator Selection: At the first stage of this project, the program would randomly select, with equal probability, either crossover or mutation. After this, it would randomly select a specific genetic operator of that category from those available. For the third stage of this project, the barrier between crossover and mutation was removed and initially each specific genetic operator has the same probability of being selected. Unlike before, this probability does not have a fixed value, it is meant to be able to increase or decrease according to its effect on the fitness of the individuals.

The evolution ends when the maximum number of generations is reached or when an individual has perfect accuracy on the training set. Until one of these conditions is

n	Number of nodes of the individual
d	Number of dimension of the individual
c	Number of classes on the dataset
s	Number of samples on the training set
s_i	Coordinates of the sample i on the output space
cc_c	Coordinates of the centroid i
cl_i	Coordinates of the centroid of the labeled class for the sample i
Acc	Overall accuracy of the individual

Table 3.3: Variables used in the fitness functions

met, the following steps will occur in a cycle: The individuals are evaluated using the chosen fitness function. They are then sorted from best to worst. The best individual is selected to be pruned and is added to the next generation, while the second best is selected through elitism. Until the next generation has 500 individuals, individuals are selected via tournament to be used in crossover or mutation. If the resulting individual from a genetic operation has a depth greater than 17, it is discarded.

3.3.2 Fitness Functions

As mentioned before, in this project we intend to replace an accuracy-based fitness function with a distance-based fitness function. To do that, we implemented and tested two distance-based fitness functions. Since these functions use the Euclidean distance, to have a fair point of comparison, we also implemented an accuracy-based fitness function that classifies a sample by associating it with the class of the closest centroid on the output space of the individual, using the Euclidean distance instead of the Mahalanobis distance. This function allows us to know if a distance-based fitness function has better results than an accuracy-based fitness function.

Here we describe each of the different fitness functions implemented. These functions use as variables the size and number of dimensions of the individual, the number of classes and samples on the training set, the coordinates of the samples on the output space, and the coordinates of the centroids of each class. These variables' description can be seen in Table 3.3.

Sigmoid:

Sigmoid functions have a few properties that are very useful for our fitness functions. They are strictly crescent, return a value in $]0, 1[$, and, for positive values, the growth never stops decreasing. This last part can be confirmed by deriving the sigmoid function. For positive values, the derivate is strictly decrescent and tends to 0. We decided to use this sigmoid function 3.1 because although all sigmoids have these properties, this sigmoid has a slower growth. From a x value onwards, the value obtained by $S(x)$ will be 1. This

happens because the program has a limited precision, implicitly rounding the value. Since this function has a smaller growth, this x value will be greater.

The range of values on the counter domain is exploited to normalize our functions and to untie individuals with the same accuracy on accuracy-based fitness functions. For a dataset with n features, the accuracy of an individual will be a value in $\{i/n, i \in \mathbb{N} \bigwedge i <= n\}$. This means that there is a $1/n$ gap between two values. This function is used to untie individuals with the same accuracy by removing from their accuracy the sigmoid of their size, divided by n.

The distance-based fitness functions use the growth of the sigmoid function to more easily separate the clusters while bringing the samples closer to their respective centroids. Since the growth of the sigmoid is higher for smaller values, using the sigmoid means that the variation in fitness will be smaller after the clusters are already separated, since the space will have large inter-cluster distances, and that as we bring the samples closer to their class centroid, the impact of moving them even closer will only increase, since the output space will have small intra-cluster distances.

$$S(x) = \frac{x}{1 + |x|} \tag{3.1}$$

Mahalanobis accuracy with size to untie: (MA)

The fitness function that was initially used by our implementation of the M3GP algorithm is given by the following expression. It states that individuals with higher accuracy have a better fitness. The sigmoid function returns a value that is used to untie the individuals with the same accuracy by giving the smaller individuals a better fitness. The smaller individuals have this benefict as by having less structure, they have less room to overfit [36].

$$\mathrm{Acc} - \frac{S(n)}{s} \tag{3.2}$$

Euclidean accuracy with size to untie: (EA)

Due to the high computational complexity of the previous fitness function, we implemented this function to be used in the second and third stages of the project. With this function, we can obtain the results way faster since we are using a distance with a linear complexity, instead of a distance with polinomial complexity. This function works in the same way as the previous expression but uses the Euclidean distance instead of the Mahalanobis distance.

Euclidean distance divided by the number of dimensions: (ED1)

A general problem with working with high dimensions is that intuition tends to fail when we are making an attempt to ”visualize” the search space. One case where intuition

fails, mentioned in [36], is that *"Most of the volume of a high-dimensional orange is in the skin, not the pulp."*. If we have an orange whose pulp radius is 95% of its total radius, in $\mathbb{R}^3$ the pulp volume will be 85.7% of the total volume of the orange, while in $\mathbb{R}^{20}$ the pulp will be only 35.8% of the total volume. Another well-known problem with high-dimensional spheres is a *"the High-Dimensional Spheres in Cubes paradox"*. Image a space in $\mathbb{R}^2$ that has a square with opposite vertices in $(1, 1)$ and $(-1, -1)$, and unit circles in each vertice. If we want to insert in this square the largest circle we can that does not touch the sphere, we have to subtract the radius of the unit circles to half of the diagonal of the square. This means that if we use the Pythagoras' Theorem we conclude that the radius of the inner circle will be $\sqrt{2} - 1$. The Pythagoras' Theorem can be extended to $\mathbb{R}^n$, allowing us to know that the diagonal of an n-dimensional cube, with side l, is $l\sqrt{n}$. This paradox starts in five dimensions. The solution to this problem is the same as before. The 5-dimensional sphere will have a radius of $\sqrt{5}-1$. Since $\sqrt{5}-1 > 1$, the inner sphere will have a radius that is larger than the side of the 5-dimensional cube.

Although ignoring these problems and hope that the classifier adapts to each is a valid approach, it is important to acknowledge that these problems exist and try to minimize their effect on the classifier. Since distances in higher dimensions tend to be higher, comparing individuals with a different number of dimensions can be something not as trivial as comparing individuals with the same number of dimension. As an attempt to minimize this problem, we try to normalize the distances across individuals with a different number of dimensions by dividing the distances by the number of dimensions.

We made this function 3.3 as an attempt to increase the distance between clusters while reducing the distance that each sample has to the centroid of its labeled class.

Since we are using the sigmoid function, after separating the cluster, the individual will not benefit from further separation. On the other hand, the benefit each sample has for getting closer to the centroid will always be increasing as the growth of the function is greater for distances closer to zero.

$$S\left(\frac{\sum_{i=1}^{c}\sum_{j=i+1}^{c} cc_i \cdot cc_j}{c \times (c-1) \times d}\right) - S\left(\frac{\sqrt{\sum_{i=1}^{s}\left(s_i \cdot cl_i\right)^2}}{d}\right) \tag{3.3}$$

Euclidean distance divided by the root of the number of dimensions: (ED2)

After the previous fitness function was tested, we noticed that the individuals tended to have the same number of dimensions as those on the Euclidean accuracy function. Although this shows that we can have an equivalent population when using a distance-based fitness function, we wanted to try another distance-based fitness function 3.4 that would have a better normalization of the distances across spaces with different numbers of dimensions. Since the distance from a point in $\mathbb{R}^n$ with all coordinates set as 1 to the origin of the space is $\sqrt{n}$, we made a slight change to the function in a new attempt

to normalize the distances between individuals with a different number of dimensions, hoping that this change would lower the number of dimensions.

$$S\left(\frac{\sum_{i=1}^{c}\sum_{j=i+1}^{c} cc_i \cdot cc_j}{c \times (c-1) \times \sqrt{d}}\right) - S\left(\sqrt{\frac{\sum_{i=1}^{s}(s_i \cdot cl_i)^2}{d}}\right) \tag{3.4}$$

3.3.3 Genetic Operators

All M3GP variants until now have worked with genetic operators that have fixed probabilities of being selected. This selection was made by first randomly selecting a category, either crossover or mutation, and then randomly selecting a specific genetic operator of this type. In this project, we try to improve the results by implementing a new solution for the selection of the genetic operators.

Before explaining how our proposal was implemented, it is necessary to remind the reader that what we intend to do is evolve the probabilities each genetic operator has of being used. The algorithm increases or reduces the probability of occurence of a genetic operator according to the effect it has on the individuals' fitness. In this implementation, each individual will evolve the probability it has of selecting each specific genetic operator.

The modifications made to the algorithm were quite simple. First, the division between crossover operators and mutation operators was ignored, leaving all the genetic operators under the same category. This means that each genetic operator has the same probability of being chosen. Since the individual is supposed to evolve the probability that each GO has of being used, it received a new attribute, an array with size equal to the number of GOs where all positions start with the value 1. The probability a genetic operator has of being selected is equal to its value divided by the sum of the value of all operators.

To do the selection of genetic operators, we implemented a roulette, weighted with the values of the individuals' probability array. This method associates a randomly generated value between 0 and the sum of the values in the array to a genetic operator. After a genetic operator is chosen and used, it creates a descendant that receives the probability array from one of its parents. When a genetic operator uses more than one individual, the same number of descendants is made and each descendant receives the probability array of one of its parents. The descendant is then evaluated using the fitness function. If the individual has a better fitness than its parents average, the value that genetic operator has in the descendants' array is increased by 0.1. If the fitness decreased, the value is decreased by 10% but never goes below 0.1. We force the value to stay above 0.1 so that the algorithm never rejects a genetic operator completely, as it may be bad at the start of

the evolution but be useful in later generations. One trivial example is the genetic operator that removes one dimension. Since the individuals are created with only one dimension, this method is useless at the first generation. However, if its probability drops too low, it may never be used on later generations where it is needed, because its value will be near 0.

After we observed the results using the five original genetic operators, five other genetic operators were implemented to test the capability the algorithm has to avoid bad genetic operators and, to test the algorithm reaction when faced with our new crossover operator. This crossover method is different from the ones the algorithm already has because it uses three individuals as parents. Besides this crossover genetic operator, we implemented four mutation genetic operators that are meant to be bad genetic operators. The crossover genetic operators one and two, and the mutation genetic operators one through three, are the genetic operators used on the original implementation of the M3GP algorithm. As they were already mentioned in 3.3.1, they will not be mentioned here. A brief description of what the newly implemented genetic operators do can be seen here:

Mutation 4 (Grow or Trim): This operator first randomly selects a node within an individual, and has a different behavior whether the node is a terminal or a non-terminal. If the node is terminal, it applies the standard GP mutation in this node. If the node is non-terminal, it applies the **Trim** method in this node.

Mutation 5 (Trim): This method first randomly selects a node within an individual, and then replaces it with a terminal node.

Mutation 6 (Chop): This method first randomly selects a dimension within the individual, and then turns the root node of this dimension into a terminal node.

Mutation 7 (Singularitree): This method first removes all of the individual's dimensions, then adds a branch, created using the Full method with depth 1, to the individual's dimensions. The result is an individual with only one node.

Crossover 3 (Swap3-dim): This method first randomly selects one dimension in each of three individuals, then it moves the first individual's dimension to the second individual, the second individual's dimension to the third individual, and the third individual's dimension to the first individual. We informally call this operator *"menage à trois"*.

Chapter 4

Implementation

4.1 Overview

This implementation is based on the standard Genetic Programming algorithm, proposed by J. R. Koza in *Genetic Programming: vol. 1, On the programming of computers by means of natural selection* (1992) [3] and is extended to the M3GP algorithm using the algorithm proposed by Munõz L., Silva S. and Trujillo L in *M3GP - Multiclass Classification with GP* (2015) [4]. The main change to the algorithm was that now each individual's instead of containing a single node, they will contain a mutable array of nodes. In addition to this, the old methods for both crossover and mutation and the fitness function had to be replaced. Some additional classes also had to be implemented in order to support some more complex fitness functions. More details about the function of each class will be given in the next section.

This implementation is meant to be used either on the terminal, by creating a runnable jar file using the ClientWekaSim as the main class, or as part of a machine learning package, Weka [6].

4.2 Java Implementation

This section contains a brief explanation of the class diagram, followed by the description of the function that each class has in our original implementation of the M3GP algorithm. The class diagram made for this implementation has been split into three parts for a better comprehension:

- Initizalition:

  The part of the class diagram referent to the initialization of the classifier can be seen in A.1. Here, we can see that ClientWekaSim first uses the class Data to read a data file and then creates a population. The M3GP class does not need to do this as the dataset is read by Weka.

- Evaluation:

  The part of the class diagram referent to the evaluation of the population's individuals can be seen in A.2. Here, we can see that the individuals (Tree) use their Nodes to calculate the output values of the samples to then use the Arrays and Matrix classes to make the prediction they need to know their accuracy.

- Evaluation:

  The part of the class diagram referent to the usage of genetic operators on the population's individuals can be seen in A.3. Here, we can see that the PopulationFunctions class uses each genetic operator type Handler to obtain new individuals. These Handlers use the NodeHandler class to make alterations to the individuals and obtain new ones.

The remainder of this section will describe the function that each class has in our original implementation of the M3GP algorithm.

- weka.classifiers.trees.M3GP

  This class can be the one used by Weka to run the classifier. It receives the dataset from Weka and will use the default values for all variables of the classifier. The user may change some of them, such as the population size and the number of generations using parameters on Weka.

  The number of variables the user can change was kept to a minimum since we decided that the user should have as little input as possible, as one of the objectives is to evolve the parameters rather than having them hardcoded or chosen by the user. The reduced number of options also has the benefit of making the classifier more user-friendly by asking only what might be critical to the user since its computer might not have the processing power to compute a large population or the time to complete many generations.

- weka.classifiers.trees.m3gp.client.ClientWekaSim

  Since initially this implementation was meant to be part of Weka and every time we changed the code we had to copy the classes to the Weka folder and recompile the project, we made this class to simulate Weka and allow us to run the classifier in a stand-alone version.

  This class contains all the parameters as static attributes so that the user/programmer may change them easily. This class has as an output a JSON file containing the results of the run. These results are the model of the best individual of each generation and the output space that is obtained by converting the dataset.

- weka.classifiers.trees.m3gp.node.Node

  The objects from this class represent nodes of a binary tree. Each Node object contain as atributes a Double value v and two Nodes l and r which will not be initialized if the Node is terminal.

  If a node has both l and r set as $null$, the node is considered terminal, otherwise, the node is non-terminal. For terminal nodes, v can either represent an index a feature of a sample or be a random number between 0 and 1, decided when the node was created. This means that $v \in \{\mathbb{N}_{<No.\,operators} \cup]0, 1[\,\}$. For non-terminal nodes, v is an index to a function on the fuction set, making $v \in \mathbb{N}_{<No.\,operators}$.

  In this project, the function set includes only the sum, subtraction, multiplication, and protected division operations. This means that a node will always return a value in $\mathbb{R}$.

- weka.classifiers.trees.m3gp.node.NodeHandler

  This class contains basic functions that allow operations with nodes. Functions such as the selection of a random node within the structure, swapping the contents of two nodes and redirecting one node to another.

- weka.classifiers.trees.m3gp.population.Population

  This class is the core of the classifier. It has a constructor that receives all the parameters, a void method to evolve the population, and a String method that receives a dataset row and returns the prediction for the classification.

  This class uses the methods from PopulationFunctions to evolve the population.

- weka.classifiers.trees.m3gp.population.Population$FitnessCalculator

  The objects of this class are only meant to allow the classifier to use multiple threads to do parallel work while evaluating the individuals. When the evaluation step of begins, the programs creates a pool using n threads. This pool is then filled with tasks. Each task is the evaluation of an individual. After the tasks are all submitted, the program calculates the fitness of the individuals in parallel, saving execution time.

- weka.classifiers.trees.m3gp.population.PopulationFunctions

  The core of the algorithm is in the class Population and it is not supposed to be changed. We made this class in order to make the alterations to the implementations easier by separating the functions related to the evolution of the population. This class contains the methods responsible for the fitness and for calling the handlers for the crossover, mutation, and pruning.

- weka.classifiers.trees.m3gp.tree.Classification

  This class contains a classification method with a switch to redirect the call to the classification method selected at the start of the run. This class was made to switch easily between the classification methods when trying new functions.

- weka.classifiers.trees.m3gp.tree.Tree

  This class' Objects represent the individuals of the population.

  Each individual contains an ArrayList of nodes that represent the dimensions of the individual, an ArrayList of centroids, and an ArrayList of the respective classes. The individuals also have attributes which might or might not be initialized depending on the fitness function and classification method used. These attributes are a Double matrix which contains the coordinates of each point from the output of the training set and the respective covariance matrix.

  Since each individual contains an ArrayList of nodes and each node outputs a value in $\mathbb{R}$, the individuals' output will be a value in $\mathbb{R}^n$.

- weka.classifiers.trees.m3gp.tree.TreeCrossoverHandler

  This class contains the method responsible for the crossover between two individuals. This method receives two parent individuals and returns two descendants. One of the available crossover methods randomly selects a dimension in each of the individuals and swaps them, while the other method randomly selects a node within each individual and swaps the node, using the crossover algorithm from standard GP.

- weka.classifiers.trees.m3gp.tree.TreeMutationHandler

  This class contains the method responsible for the mutation of an individual. This method receives an individual and picks one of the methods available for the individual with an equal chance. It can mutate one dimension, using the same process as the standard GP algorithm, create a new dimension by adding a new node to the individuals root and mutating it or remove a randomly selected dimension. The method only removes a dimension if the individual has at least two dimensions.

- weka.classifiers.trees.m3gp.tree.TreeGeneticOperatorHandler

  We implemented his class to replace the Mutation and Crossover handlers for the third stage of the project. This class contains all the genetic operator methods from the initial algorithm and a few other methods, mentioned at section 3.3.3.

- weka.classifiers.trees.m3gp.tree.TreePruningHandler

  This class contains the method responsible for pruning an individual. For each dimension the individual has, this method removes it and compares the resulting

individuals' fitness to the original. If the fitness improves, this dimension is permanently removed from the individual. This allows the removal of dimensions that are prejudicial to the individuals' fitness.

- weka.classifiers.trees.m3gp.util.Arrays

  This class contains methods that operate with Arrays that were found useful to implement. The only two methods that are worth mentioning here are the method that calculates the Mahalanobis[12] distance and the one that makes the merge sort. The implementation of the merge sort is used to sort the individuals of the population by the values in the array containing the fitness of each individual.

- weka.classifiers.trees.m3gp.util.Data

  This class is meant to be used only by the WekaSimClient. It has only one method which reads a .csv file, stores the samples and features of the dataset in a Double matrix, and the labeled classes of the samples in a String Array. This method returns an Array containing the Double matrix on the first position the String Array on the second position.

- weka.classifiers.trees.m3gp.util.Matrix

  This class contains a few methods involving matrices. The more relevant methods are those that calculate the Moore-Penrose inverse matrix [5], the covariance matrix, and the inverse matrix.

Chapter 5

Results

5.1 Implementation of our M3GP

In order to validate our implementation, several runs using the same datasets and parameters as those in *M3GP - Multiclass Classification with GP* (2015) [4] were made. The results obtained will be analyzed in the next two subsections. The boxplot graphs displayed in this project were made using R [41] and the plots displayed were made using the Matplotlib library [42] for Python 3.

5.1.1 Comparison of results

In Table 5.1, we can see the comparison made between the original algorithm and the results obtained with our implementation. These results are all related to the best individuals of the last generation. The table contains the median values of 30 runs for the training and test fitness, the number of nodes of the individual and its number of dimensions, also indicating the minimum and maximum number of dimensions, observed on the best individual at the end of each run.

Despite having followed the M3GP specifications in [4], the number of nodes and dimensions we observed were outside the expected range of values in most of the datasets. This may have been due to the original M3GP being implemented over the GPLAB toolbox [39] which contains additional bloat control [37, 38] measures set by default.

In terms of accuracy, the medians are similar in most cases, but a deeper analysis was performed and two boxplots were drawn, one for training accuracy (Figure B.2) and one for test accuracy (Figure B.3). Once again, some differences were detected, both in training and test accuracy. As the results we achieved were not necessarily worse than the original ones, we accepted we could use this implementation in the next stages of the project.

	HRT	IM-3	IM-10	M-L	SEG	VOW	WAV	YST
Training fitness								
Original M3GP	0.947	0.996	0.930	1.000	0.981	1.000	0.907	0.685
Our implementation	0.942	0.991	0.942	1.000	0.979	1.000	0.885	0.719
Test fitness								
Original M3GP	0.790	0.954	0.910	0.571	0.956	0.938	0.843	0.562
Our implementation	0.790	0.948	0.921	0.532	0.958	0.919	0.849	0.549
Number of Nodes								
Original M3GP	110	66	239	13	111	53	71	274
Our implementation	219	57	495	644	390	229	49	511
Nr. of dimensions								
Original M3GP	12 (1-17)	5 (2-8)	12 (11-16)	12 (10-13)	11 (5-21)	20 (16-20)	31 (29-37)	13 (11-18)
Our implementation	10 (6-19)	5 (3-9)	29 (11-40)	13 (11-15)	15 (6-22)	20 (18-24)	20 (17-22)	25 (12-37)

Table 5.1: Comparison of results between the original implementation of M3GP and our implementation

5.1.2 Evolution of the population

The values displayed in Table 5.2 are the median of the number of nodes, and dimensions, within the best individuals of the 15^{th}, 25^{th}, 50^{th}, and 100^{th} generations. This table also contains the median of the number of nodes per dimension of the best individual, as we wanted to know how the size of the dimensions would grow over the generations.

These results indicate that the populations trying to learn datasets with more classes, such as the IM-10, M-L, VOW, and YST, have a greater initial growth on the number of dimensions. This suggests that it is highly beneficial to the individuals to explore spaces in high dimensions. On the other hand, populations learning datasets such as HRT, SEG, and WAV, initially explore spaces with a lower number of dimensions and then tend to increase the number of dimensions with the passing of the generations. This might mean that although the population can initially evolve the individuals while maintaining them with few dimensions, at some point it needs to increase the number of dimensions to learn the datasets. With the IM-3 dataset, the population starts by exploring spaces with few dimensions and continues this way, not finding benefits in exploring spaces in higher dimensions. It's worth mentioning that although the individuals can stop their training by achieving perfect training accuracy, this is not the case. As we can see in Table 5.1, the median of the training accuracy for the IM-3 dataset is not 1, meaning that at least in half the runs, the state of perfect training accuracy was not reached.

Another conclusion we take from Table 5.2 is that with the initial generations, we might be able to know if a dataset will need complex or simple trees to be learned since with the exception of the SEG and VOW datasets, the number of nodes per dimension is on the same order of magnitude.

In previous work [8], the M-L dataset showed a peculiar behaviour when compared with other datasets. The M2GP was tested several times on this dataset using populations with 1 to 20 dimensions. Starting from the populations that had 14 dimensions onward, all populations showed lower training accuracy than the populations that had 13 dimensions. This may justify why the M-L populations stop growing after reaching 13 generations.

	HRT	IM-3	IM-10	M-L	SEG	VOW	WAV	YST
No. of Nodes								
15th Generation	133	61	105	644	65	48	13	150
25th Generation	168	69	167	644	118	97	10	224
50th Generation	202	58	324	644	210	216	16	394
100th Generation	219	57	495	644	390	229	49	511
No. of Dimensions								
15th Generation	5	4	9	11	6	8	6	10
25th Generation	6	5	12	13	8	12	8	13
50th Generation	8	5	20	13	11	20	13	19
100th Generation	10	5	29	13	15	20	20	25
No. of Nodes per Dimension								
15th Generation	26.6	15.3	11.7	58.5	10.8	6.00	2.17	15.0
25th Generation	28.0	13.8	13.9	49.5	14.8	8.08	1.25	17.2
50th Generation	25.3	11.6	16.2	49.5	19.1	10.8	1.23	20.7
100th Generation	21.9	11.4	17.1	49.5	26.0	11.5	2.45	20.4

Table 5.2: Evolution of the number of nodes and dimensions over the generations for our implementation

5.2 Fitness Functions

For the second stage of this project, we wanted to replace the accuracy-based fitness function of M3GP with a distance-based fitness function, with the objective of promoting a smoother and faster evolution. As one of the motivations for this work was to reduce the excessive computational complexity of the original fitness function, we decided to abandon the Mahalanobis distance on this stage of the project, and use only the Euclidean distance. Since previous work in the M2GP [8] algorithm compared these two distances for classification and showed that using the Mahalanobis distance provided better results, instead of using the results obtained on the previous stage of this project, we compared the new distance-based fitness function results with the results obtained when using an accuracy-based fitness function that uses the Euclidean distance (EA). Both new distance-based fitness functions have the objective of separating the class centroids on the output space of the individual while bringing their respective samples closer to the centroid.

As mentioned before, in 3.3.2, two distance-based fitness functions were implemented. Since distances tend to be greater in spaces of higher dimensions, an attempt was made to normalize the distances in order to compare individuals with a different number of dimensions. This was made in the first sigmoid fitness function (ED1) by dividing the distances by the number of dimensions, and in the second sigmoid fitness function (ED2) by dividing the distances by the root of the number of dimensions.

5.2.1 Comparison of results

Of the two fitness functions tested, the ED2-FF was at a clear loss since it did not allow the population to produce a geometric space where the samples could be correctly classified. On the other hand, the ED1-FF produced results extremely similar to the ones obtained using the EA-FF, (even after a re-run with both functions).

Looking at Table 5.3, one can see that the populations using the ED2-FF kept their

number of dimensions really low while keeping each dimension much larger than whole individuals from populations using EA-FF and ED1-FF. This can be caused by the fitness function not being able to normalize the individuals across different dimensions properly, punishing too hard individuals that attempt to obtain more dimensions. Although the individuals maintained a very low number of dimensions, they try to separate the clusters with very complex functions.

Discarding the ED2-FF, we tried to identify any significant differences between the results obtained with the EA-FF and the ED1-FF. We compared the number of dimensions Figure B.1, the training and test accuracy on all datasets in the Figures B.4 and B.5.

The comparison of the results obtained by the populations using the EA-FF and the populations using the ED1-FF was made using the Kruskal-Wallis test to obtain the results' p-value, displayed in Table 5.4. Adopting a significance level of 0.01, these results show no significant differences with the exception of the IM-10 dataset that has borderline significantly worse results when using the ED1-FF.

When comparing the plot graphs made to analyse the evolution of the populations learning each datasets, we could see that the evolution of the fitness was quite similar in all datasets, having plots like those on Figures 5.1 and 5.2. The only exceptions were the HRT and MCD-3 populations, that had a bigger gap between the lines ploted, as can be seen in the Figures 5.3 and 5.4.

	HRT	IM-3	IM-10	M-L	SEG	VOW	WAV	YST
Training fitness								
EA-FF	0.894	0.940	0.829	0.743	0.923	0.719	0.842	0.591
ED1-FF	0.889	0.938	0.822	0.751	0.926	0.721	0.845	0.598
ED2-FF	0.569	0.324	0.111	0.116	0.150	0.104	0.333	0.018
Test fitness								
EA-FF	0.815	0.887	0.823	0.560	0.908	0.635	0.833	0.555
ED1-FF	0.802	0.897	0.812	0.573	0.914	0.628	0.835	0.552
ED2-FF	0.549	0.320	0.108	0.092	0.145	0.091	0.333	0.011
No. of Nodes								
EA-FF	95	46	264	839	417	120	18	245
ED1-FF	99	50	215	845	412	133	16	236
ED2-FF	836	569	555	1302	1065	609	1281	370
No. of dimensions								
EA-FF	5 (2-7)	3 (1-8)	12 (7-12)	14 (9-17)	13 (7-19)	15 (10-20)	14 (11-19)	12 (8-16)
ED1-FF	5 (1-8)	3 (1-11)	12 (6-15)	13 (8-18)	11 (10-17)	15 (12-19)	14 (10-17)	13 (10-17)
ED2-FF	1 (1-2)	1 (1-2)	1 (1-2)	1 (1-2)	1 (1-2)	1 (1-2)	1 (1-2)	1 (1-2)

Table 5.3: Comparison of results between the different fitness functions

	HRT	IM-3	IM-10	M-L	SEG	VOW	WAV	YST
EA vs ED1								
Training	0.4982	0.9586	0.0144	0.7058	0.4824	0.8649	0.0976	0.4731
Test	0.6614	0.8009	0.0087	<u>0.1196</u>	0.1758	0.6411	0.3706	0.4201

Table 5.4: p-values obtained by comparing the results from the populations learning the datasets using the EA-FF and the ED1-FF. The underline is used to symbolize that the EA had a significantly better result than ED1.

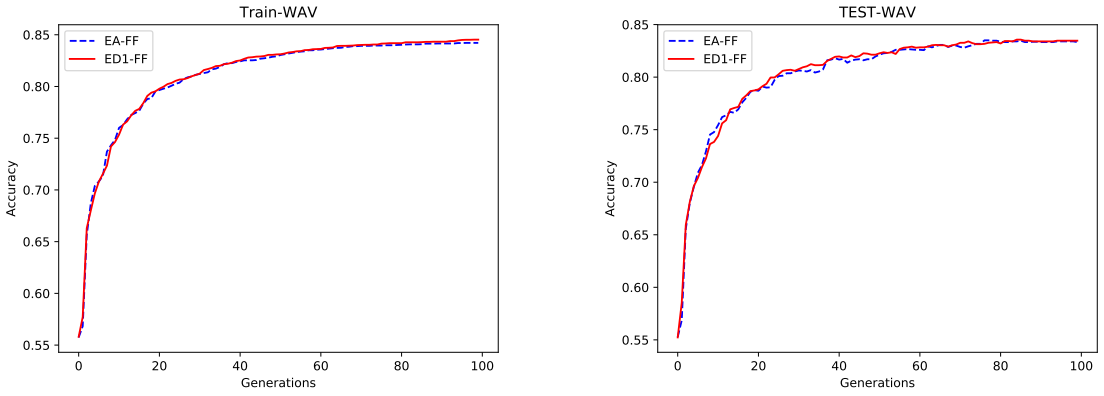

Figure 5.1: Evolution of training(left) and test(right) accuracies of the EA-FF and ED1-FF populations when learning the WAV dataset.

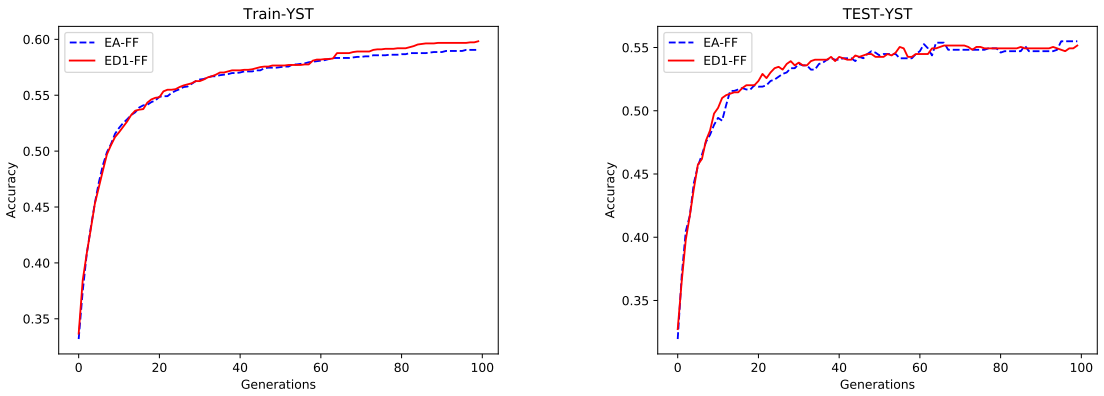

Figure 5.2: Evolution of training(left) and test(right) accuracies of the EA-FF and ED1-FF populations when learning the YST dataset.

5.2.2 Evolution of the population

In order to study the evolution of populations using the EA fitness function and populations using the ED1 fitness functions, a table containing the number of nodes, the number of dimensions, and the number of nodes per dimension on the 15^{th}, 25^{th}, 50^{th}, and 100^{th} generations was used. The information about the EA-FF population can be seen in Table 5.5, and the information about the ED1-FF can be seen in Table 5.6.

The results displayed in these two tables show that when using the Euclidean distance, the initial benefit of exploring spaces in higher dimensions is lower than when the Mahalanobis distance is used. This removes the initial difference of the number of dimensions across the datasets that could be seen when we were using the Mahalanobis distance in Table 5.2. One thing that was maintained with this change was the order of magnitude of the size of each dimension. Once again, we can see that in most datasets, the size of each dimension is maintained over the generations.

After analyzing these tables, we concluded that the evolution of the population using

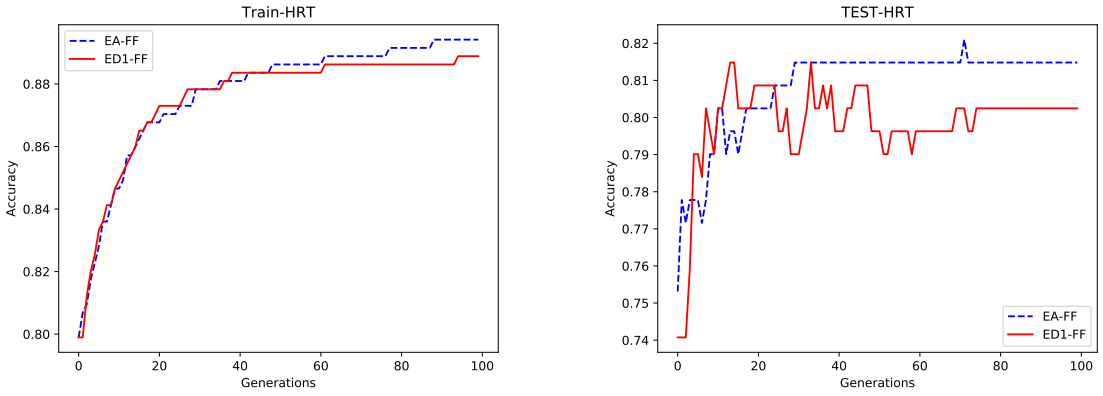

Figure 5.3: Evolution of training(left) and test(right) accuracies of the EA-FF and ED1-FF populations when learning the HRT dataset.

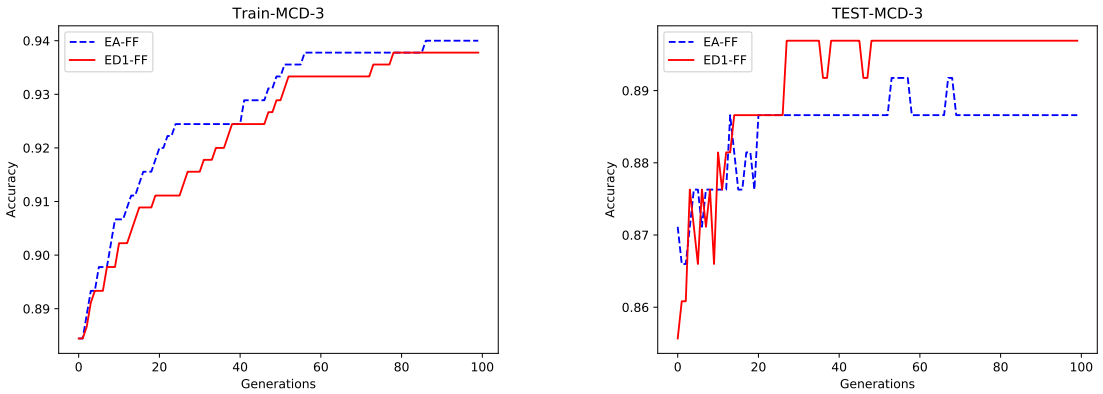

Figure 5.4: Evolution of training(left) and test(right) accuracies of the EA-FF and ED1-FF populations when learning the MCD3 dataset.

these two fitness functions is quite similar. Even the average values over all datasets are almost the same, as shown in Table 5.7. The only difference that might be worth mentioning is the average number of nodes per dimensions on the 15^{th} generation. The population using the distance-based fitness function managed to have an average of 294.3 nodes per dimension while the population using an accuracy-based fitness function only had 218.9 nodes per dimension. This suggests that the individuals using a distance-based fitness function may try to initially learn the dataset with more complex dimensions.

5.3 Genetic Operators

In the third stage of the project, we changed the method for the selection of genetic operators from one that uses a fixed probability for each genetic operator to one that tries to evolve the probability of selection that each genetic operator has. To do so, we removed the boundary between mutation and crossover genetic operators, giving all genetic opera-

	HRT	IM-3	IM-10	M-L	SEG	VOW	WAV	YST
No. of Nodes								
15th Generation	54	44	51	260	106	14	5	95
25th Generation	88	50	85	384	160	25	6	133
50th Generation	104	40	147	587	262	58	10	183
100th Generation	95	46	264	839	417	120	18	245
No. of Dimensions								
15th Generation	3	2	4	5	5	6	5	6
25th Generation	4	2	7	6	6	7	6	8
50th Generation	4	3	9	9	9	10	9	10
100th Generation	5	3	12	14	13	15	14	12
No. of Nodes per Dimension								
15th Generation	18.0	22.0	12.8	52.0	21.2	2.33	1.00	15.8
25th Generation	22.0	25.0	12.1	64.0	26.6	3.57	1.00	16.6
50th Generation	26.0	13.3	16.3	65.2	29.1	5.80	1.11	18.3
100th Generation	19.0	15.3	22.0	69.9	32.1	8.00	1.29	20.4

Table 5.5: Evolution of the number of nodes and dimensions over the generations for the EA-FF

	HRT	IM-3	IM-10	M-L	SEG	VOW	WAV	YST
No. of Nodes								
15th Generation	77	27	39	272	83	13	5	91
25th Generation	86	29	65	401	152	29	6	125
50th Generation	109	43	128	560	279	63	10	175
100th Generation	99	50	215	845	412	133	16	236
No. of Dimensions								
15th Generation	3	2	5	5	5	6	4	7
25th Generation	3	2	6	7	6	8	6	8
50th Generation	4	3	9	10	9	11	10	10
100th Generation	5	3	12	13	11	15	14	13
No. of Nodes per Dimension								
15th Generation	25.7	13.5	7.80	54.4	16.6	2.17	1.25	13.0
25th Generation	28.6	14.5	10.8	57.3	25.3	3.63	1.00	15.6
50th Generation	27.3	14.3	14.2	56.0	31.0	5.73	1.00	17.5
100th Generation	19.8	16.7	17.9	65.0	37.5	8.87	1.14	18.2

Table 5.6: Evolution of the number of nodes and dimensions over the generations for the ED1-FF

tors an equal chance of being selected. This probability then evolves with each individual, increasing everytime the operator is used and creates a descendant with a better fitness, otherwise, the probability that genetic operator has of being used is reduced. As a second sub-stage, five other genetic operators were implemented and added to the algorithm. The information in this paragraph can be further reviewed in section 3.3.3.

Since we obtained no relevant differences between using the EA-FF or using the ED1-FF, both of them were tested in this stage of the project as another attempt to find any difference in their evolution. Each fitness function was used twice, once when the five original genetic operators are used, and again when the five newly implemented genetic operators are also used. Unlike the two previous sections, this section will include a third subsection where the evolution of the genetic operators will be studied.

	Dimensions	Nodes	Nodes per dimension
EA-FF			
15th Generation	4.50	78.63	218.9
100th Generation	11.0	255.5	339.5
ED1-FF			
15th Generation	4.63	77.13	294.3
100th Generation	10.8	250.8	345.3

Table 5.7: Comparison between the average number of nodes, dimensions, and nodes per dimension of using the EA-FF and the ED1-FF, over all the datasets

5.3.1 Comparison of results

The results obtained in this stage of the project once again show a very similar evolution between the populations using the accuracy-based or distance-based fitness functions under the same conditions. Therefore, the results obtained from using the ED1-FF will not be compared with the results obtained from using the EA-FF. We will however compare the results that each population had in stage 2, displayed in Table 5.3, with those obtained in this stage when using five and ten GOs with adaptable probabilities.

One observation we can make when comparing the results obtained in this stage and displayed in Table 5.8, with the ones obtained on the second stage of the project, is that using our method for the evolution of genetic operator probabilities tends to create individuals with a higher number of dimensions. Another interesting observation we can make is that when the five additional genetic operators are used, the accuracy remains the same in most cases, while the individuals have a reduced number of nodes and dimensions. Over this stage, the YST populations have shown to, in most cases, have a different behavior than other datasets. This seems to indicate that the YST dataset is more difficult to learn.

For a deeper evalutation of the results obtained in this section, we used some boxplot graphs and the Kruskal-Wallis test to obtain the results' p-values when comparing the original EA and ED1 populations to their respective counterparts with five GOs with adaptable probabilities, and the populations using five GOs with adaptable probabilities with the populations using ten GOs. The p-values obtained can be seen in 5.9. The training accuracy boxplot graph for HRT and MCD-3 can be seen in Figure B.6, MCD-10 and M-L in Figure B.7, SEG and WAV in Figure B.8, and VOW and YST in Figure B.9. Their counterparts for the test set can be seen, respectively, in Figures B.10, B.11, B.12, and B.13.

When comparing the results obtained by having GOs with adaptable probabilities, the outcome was quite positive. When the population using the EA-FF used the adaptable probabilities, the training results were significantly better in four datasets, when compared to not adapting the probabilities. The test results were only significantly better in M-L and borderline better in VOW. The populations using the ED1-FF had a similar improvement when they started to use the adaptable GOs' probabilities. The training results from four

datasets were significantly better. The test results were significantly better when the population was classifying the MCD10 dataset and borderline better when it was classifying the VOW dataset.

When comparing the results obtained from using the five original GOs with adaptable probabilities and using ten GOs with adaptable probabilities, we can conclude that there were no significicant differences in the results with the exception of the population using the EA-FF to learn the VOW dataset and the populations using the ED1-FF to learn the M-L dataset. Both these populations had significantly worse results in the training set of the respective datasets.

the populations using the EA-FF to learn the VOW dataset and the populations using the ED1-FF have a significatly worse training accuracy, there are no significant differences on the test accuracy, in any dataset.

	HRT	IM-3	IM-10	M-L	SEG	VOW	WAV	YST
Training fitness								
EA-FF								
- 5 Gen. Op.	0.894	0.938	0.838	0.793	0.933	0.750	0.850	0.611
- 10 Gen. Op.	0.894	0.929	0.829	0.787	0.934	0.735	0.849	0.602
ED1-FF								
- 5 Gen. Op.	0.889	0.936	0.831	0.805	0.932	0.758	0.850	0.606
- 10 Gen. Op.	0.884	0.938	0.829	0.781	0.932	0.750	0.851	0.604
Test fitness								
EA-FF								
- 5 Gen. Op.	0.827	0.887	0.827	0.610	0.915	0.667	0.837	0.565
- 10 Gen. Op.	0.809	0.887	0.820	0.601	0.921	0.663	0.837	0.561
ED1-FF								
- 5 Gen. Op.	0.809	0.876	0.824	0.606	0.916	0.653	0.839	0.561
- 10 Gen. Op.	0.809	0.876	0.820	0.606	0.916	0.657	0.837	0.553
No. of Nodes								
EA-FF								
- 5 Gen. Op.	159	60	366	1060	509	200	22	408
- 10 Gen. Op.	58	35	263	754	348	158	18	243
ED1-FF								
- 5 Gen. Op.	158	62	378	1176	629	224	20	364
- 10 Gen. Op.	62	34	274	763	300	161	18	278
No. of Dim.								
EA-FF								
- 5 Gen. Op.	7 (1-14)	3 (1-16)	18 (10-25)	19 (15-30)	20 (5-29)	24 (16-38)	17 (14-21)	22 (12-43)
- 10 Gen. Op.	4.5 (1-9)	2 (1-6)	17 (10-24)	15 (11-23)	16 (5-24)	20 (13-26)	16 (14-21)	17 (11-24)
ED1-FF								
- 5 Gen. Op.	5.5 (3-20)	3 (1-9)	21 (9-34)	21 (15-29)	19 (11-30)	24 (13-35)	18 (14-25)	24 (12-32)
- 10 Gen. Op.	4 (1-11)	2 (1-8)	15 (9-23)	16 (10-30)	15 (9-26)	20 (16-29)	17 (13-21)	18 (9-30)

Table 5.8: Comparison of results between the Euclidean Accuracy fitness function and the first sigmoid fitness function for both five and ten genetic operators (GO)

5.3.2 Evolution of the population

The results obtained suggest that by evolving the operator probabilities over time, the average number of nodes per dimension is not as stable as on the previous stages of this project. This change may come from an oscillation on the probabilities of the operators that mutate the individuals. The probability of using mutation operators change

	HRT	IM-3	IM-10	M-L	SEG	VOW	WAV	YST
EA vs EA(5)								
Training	0.6072	0.7108	0.0537	**1.8890**$^{-6}$	0.0443	**3.0310**$^{-8}$	**1.7790**$^{-6}$	**3.640**$^{-4}$
Test	0.3931	0.6088	0.3439	**9.2800**$^{-4}$	0.0688	**0.0057**	0.0308	0.0821
ED1 vs ED1(5)								
Training	0.9703	0.6512	**6.0298**$^{-4}$	**1.2960**$^{-9}$	0.1102	**2.0820**$^{-8}$	**9.0990**$^{-6}$	0.0179
Test	0.6667	0.3024	**0.0023**	0.2416	0.5837	0.0111	0.0434	0.1084
EA(5) vs EA(10)								
Training	0.5464	0.0841	0.1171	0.7669	0.7900	4.528^{-4}	0.8417	0.0351
Test	0.1500	0.5673	0.4687	0.1752	0.3037	0.7003	0.7170	0.4032
ED1(5) vs ED1(10)								
Training	0.7041	0.9882	0.2009	0.0002	0.3475	0.5840	0.9352	0.2308
Test	0.6607	1.0000	0.2804	0.7840	0.8532	0.4197	0.6625	0.6412

Table 5.9: p-values obtained by comparing the results from the populations learning the datasets using the EA-FF and the ED1-FF, with their respective counter part with 5 GOs with adaptable probabilities and these counterparts with the 10 GOs counterparts. The underline is used to symbolize that the first population has a significantly better result and bold symbolizes a worse result.

over the generations, making the population grow and shrink in their number of nodes and dimensions when it is needed, instead of having the tendency to keep growing over the generations. Comparing the results obtained when using the EA-FF with five and with ten genetic operators, we can see that when we use more genetic operators, the population is smaller, both on the number of node per individual and on the number of dimensions per individual. Since most of the new genetic operators try to destroy the structure of the individual, this is expectable. Although we are adapting the probabilities that each genetic operator has of being selected, we are still using many operators that try to shrink the individuals. The results obtained for the number of nodes, dimensions, and the number of nodes per dimension, can be seen in 5.10 for the Euclidean Accuracy runs with five genetic operators and in 5.11 for the runs with ten genetic operators.

	HRT	IM-3	IM-10	M-L	SEG	VOW	WAV	YST
No. of Nodes								
15th Generation	103	29	65	316	129	18	5	106
25th Generation	145	35	119	460	178	42	7	163
50th Generation	166	39	214	733	317	106	13	239
100th Generation	159	60	366	1060	509	200	22	408
No. of Dimensions								
15th Generation	4 (1-7)	1 (1-4)	6 (4-10)	6 (4-9)	6 (3-10)	7 (5-10)	5 (3-6)	8.5 (5-13)
25th Generation	4.5 (1-9)	2 (1-11)	8 (5-12)	9 (6-14)	8 (5-12)	9 (6-12)	7 (6-11)	10 (6-20)
50th Generation	6 (1-13)	2 (1-9)	12 (7-19)	14 (9-23)	13 (5-19)	16 (12-29)	12 (8-15)	15 (9-25)
100th Generation	7 (1-14)	3 (1-16)	18 (10-25)	19 (15-30)	20 (5-29)	24 (16-38)	17 (14-21)	22 (12-43)
Nodes per Dim.								
15th Generation	25.8	29.0	10.8	52.7	21.5	2.57	1.00	12.5
25th Generation	32.2	17.5	14.9	51.1	22.3	4.67	1.00	16.3
50th Generation	27.7	19.5	17.8	52.4	24.4	6.63	1.08	15.9
100th Generation	22.7	20.0	20.3	55.8	25.5	8.33	1.29	18.5

Table 5.10: Evolution of the number of nodes, dimensions, and nodes per dimension over the generations for the EA-FF with five genetic operators

	HRT	IM-3	IM-10	M-L	SEG	VOW	WAV	YST
No. of Nodes								
15th Generation	46	25	54	267	81	11	4	92
25th Generation	46	28	84	363	122	32	7	124
50th Generation	50	34	158	555	219	84	11	189
100th Generation	58	35	263	754	348	158	18	243
No. of Dimensions								
15th Generation	2 (1-4)	1 (1-3)	5 (3-9)	5 (4-7)	5 (3-9)	6 (5-9)	4 (4-6)	7.5 (4-10)
25th Generation	3 (1-6)	1 (1-3)	7 (4-11)	7 (4-15)	7 (4-12)	9 (7-15)	6 (5-8)	10 (6-14)
50th Generation	4 (1-10)	1 (1-5)	11 (5-14)	11.5 (7-19)	10 (5-17)	14 (10-17)	11 (8-15)	13.5 (9-19)
100th Generation	4.5 (1-9)	2 (1-6)	17 (10-24)	15 (11-23)	16 (5-24)	20 (13-26)	16 (14-21)	17 (11-24)
Nodes per Dim.								
15th Generation	23.0	25.0	10.8	53.4	16.2	1.83	1.00	12.3
25th Generation	15.3	28.0	12.0	51.9	17.4	3.56	1.17	12.4
50th Generation	12.5	34.0	14.36	48.3	21.9	6.00	1.00	14.0
100th Generation	12.9	17.5	15.5	50.3	21.8	7.90	1.13	14.3

Table 5.11: Evolution of the number of nodes, dimensions, and nodes per dimension over the generations for the EA-FF with ten genetic operators

5.3.3 Evolution of the operator probabilities

Here, we will give a brief commentary about the evolution of the probabilities that each genetic operator (GO) had, on average, on the 15^{th}, 25^{th}, 50^{th}, and 100^{th} generations of each dataset. First, we will comment on this evolution in the case where we only used the five original genetic operators, and then we will comment the case with ten genetic operators.

The tables containing these values are on the appendices. For the runs using five genetic operators, the table for the EA-FF is found in 5.12, and the table for the ED1-FF is found in 5.13. Their respective tables for ten genetic operators can be found in 5.15, and 5.17. For further analysis of the results, these tables also contain the standard deviation of the probabilities in each generations.

The five original genetic operators

For this sub-stage, we ran the M3GP with the five original operators, with each GO having an initial probability of 20%. Before commenting on the evolution of each genetic operator's probability of being selected, it is worth to mention that in most cases, the standard deviation of the probability was a small fraction of the probability. This means that the probabilities had some consistency across the runs, giving us some ground to be commented on.

Crossover 1 (St-XO):

In all the datasets tested, this probability kept increasing over time, achieving probabilities over 49% on seven of the eight tested datasets on the 100^{th} generation, on both the fitness functions, being YST the only dataset that does not show this behavior. It has however above average probability on this GO, 29% with the EA-FF and 37% with ED1-FF. This might suggest that as the generation pass, the advantages of using the **St-XO** GO

will increase.

Crossover 2 (Swap-dim): With both fitness functions, the probability of this GO being selected decreased over time in all datasets. The main difference was that with ED1-FF, the probability this GO has of being selected was never above 7%, while with the EA-FF, on early stages, in seven of the eight datasets, this operator has a probability between 16%, and 25%. These results indicate that when working with this distance-based fitness function, swapping dimensions between individuals probably will not result in a fitness improvement.

Mutation 1 (St-Mut): This is the only case where the standard deviation value is near the value of the probability of selection of the GO. Although this value is, with the exception of HRT, always below 10%, since the standard deviation has high values, this means that the need for this genetic operator varies with the population. This operator replaces a randomly selected branch with a new branch, created using the Grow method. The individual is improved if the replaced branch is better than the selected branch, or spoiled if the replacement branch is useless. This GO's probability is one of the two lowest in all datasets, ergo we conclude that the most common scenario is the spoiling of the individual. The likelihood of the individual being spoiled will only increase with the passing of the generations as the individuals are allegedly more well structured.

Mutation 2 (Add-dim): As already noticed in an earlier stage of this project, populations that are learning datasets with more classes have a greater dimensional growth on early stages of the evolution. The results obtained show this by maintaining low probabilities of selection of this GO on the datasets with a lower number of classes, and higher probabilityies for datasets with a higher number of classes. It's worth mentioning that IM-10, M-L, and YST, although having a higher number of classes, reach a number of dimensions that allow them to learn the dataset quickly, and then they try to use other GOs. This can be seen by the drop of the probability in both cases.

Mutation 3 (Rem-dim): Unexpectedly, in most datasets, the probability of this GO does not increase with the passing of the generations. This may be caused by the automatic adaptation of the GO probabilities. Since the populations are learning when to use the **Add-dim** GO, they may be learning when not to use it, making this genetic operator less beneficial as the need to remove dimensions is reduces. Although the YST populations have a low probability of selection this GO, they try to increase its probability. This may suggest that the increase of a GO probability when it benefits the individual is too big, or that there should exist a limit for a GO's probability. The probability that the YST populations give to the **Add-dim** GO seems to be too high for it to be reduced and allow the population to lose dimensions.

An interesting result we observe when we look at the evolution of the probability of each genetic operator over the generation is that the evolution of the GO over time seems similar when using different fitness functions. This can be seen in the Figures 5.5 and 5.6.

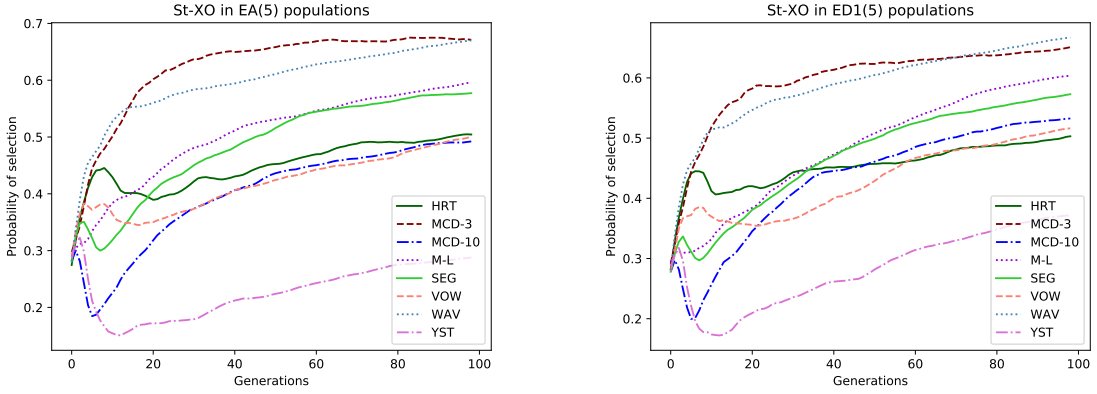

Figure 5.5: Evolution of probability of selection of the **St-XO** GO over the generations in all dataset, using the EA-FF(left) and the ED1-FF(right), using the five original GOs.

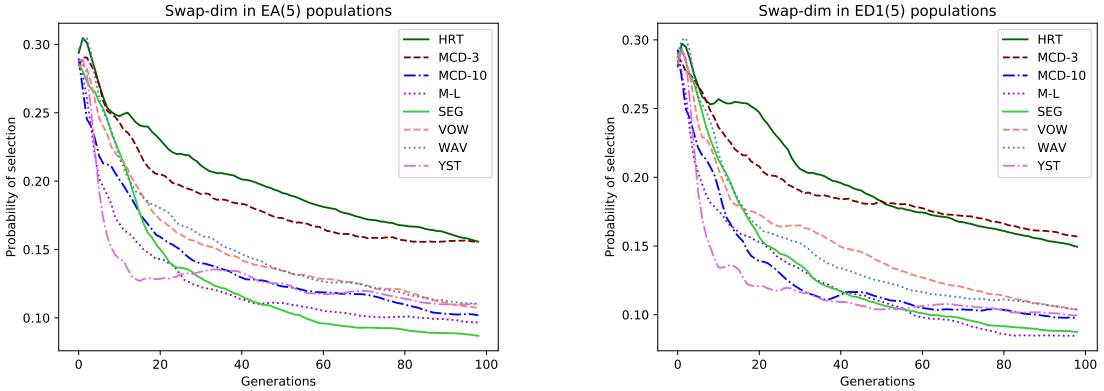

Figure 5.6: Evolution of probability of selection of the **Swap-dim** GO over the generations in all dataset, using the EA-FF(left) and the ED1-FF(right), using the five original GOs.

Another interesting result is that, although the probabilities each GO has of being selected greatly vary from dataset to dataset, the shape of the line is somehow similar across datasets. For example, in Figure 5.5, we can see that after the 20^{th} generation, all populations increase the probability of selection of the **St-XO** smoothly. Looking at Figures 5.6 and 5.7 we can see that the **Add-dim** GO usually has a great increase on the selection probability and the has a smooth decrease over time, like the **Swap-dim** GO. Lastly, looking at Figure 5.7, we can see that both the **St-mut** and the **Rem-dim** GOs initially have a great decrease on the selection probability and then, the most common scenario is a stabilization of the probability.

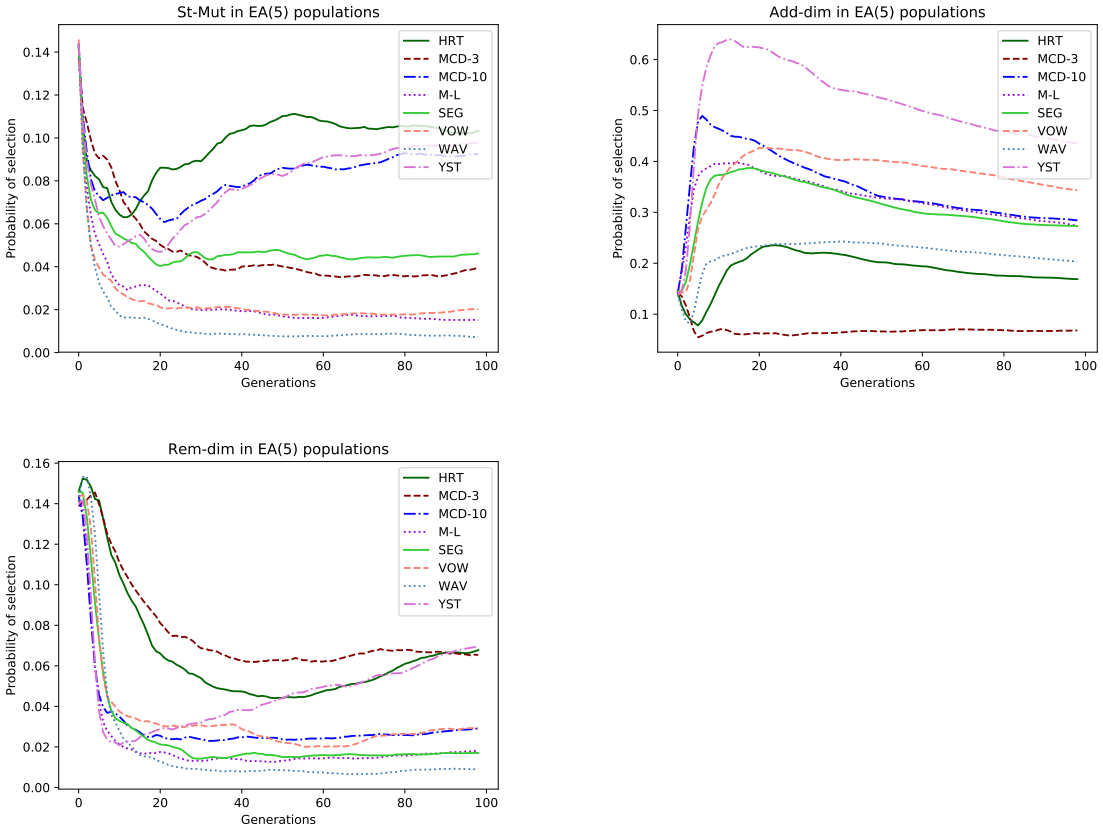

Figure 5.7: Evolution of probability of selection of the **St-Mut**(left), the **Add-dim**(right), and the **Rem-dim**(bottom) GOs over the generations in all dataset, using the EA-FF and the five original GOs.

	St-XO	Swap-dim	St-Mut	Add-dim	Rem-dim
HRT					
15th Generation	0.3998±0.1279	0.2470±0.0554	0.0639±0.0443	0.1976±0.1293	0.0915±0.0379
25th Generation	0.3919±0.1638	0.2225±0.0631	0.0850±0.0701	0.2379±0.1290	0.0624±0.0381
50th Generation	0.4452±0.1610	0.1947±0.0448	0.1092±0.1024	0.2065±0.1045	0.0441±0.0251
100th Generation	0.4995±0.1837	0.1567±0.0370	0.1049±0.0956	0.1706±0.0898	0.0681±0.0535
IM-3					
15th Generation	0.5357±0.1275	0.2324±0.0566	0.0646±0.0301	0.0672±0.0806	0.1000±0.0320
25th Generation	0.6142±0.1428	0.1994±0.0671	0.0473±0.0280	0.0646±0.1039	0.0746±0.0323
50th Generation	0.6517±0.1630	0.1757±0.0654	0.0417±0.0328	0.0683±0.0947	0.0626±0.0329
100th Generation	0.6654±0.1463	0.1573±0.0446	0.0407±0.0374	0.0717±0.0873	0.0649±0.0480
IM-10					
15th Generation	0.2599±0.0956	0.1876±0.0527	0.0728±0.0544	0.4502±0.0831	0.0294±0.0202
25th Generation	0.3390±0.1011	0.1538±0.0555	0.0620±0.0501	0.4214±0.0817	0.0238±0.0197
50th Generation	0.4303±0.1023	0.1249±0.0439	0.0841±0.0762	0.3361±0.0679	0.0246±0.0194
100th Generation	0.4923±0.1019	0.1021±0.0311	0.0917±0.0708	0.2843±0.0646	0.0296±0.0251
M-L					
15th Generation	0.3934±0.0831	0.1601±0.0353	0.0306±0.0202	0.3971±0.0683	0.0188±0.0084
25th Generation	0.4482±0.0721	0.1390±0.0363	0.0242±0.0194	0.3726±0.0630	0.0161±0.0127
50th Generation	0.5276±0.0806	0.1115±0.0260	0.0178±0.0163	0.3305±0.0634	0.0126±0.0112
100th Generation	0.5962±0.0744	0.0969±0.0262	0.0149±0.0156	0.2739±0.0592	0.0181±0.0159
SEG					
15th Generation	0.3449±0.0687	0.1957±0.0385	0.0510±0.0270	0.3792±0.0644	0.0292±0.0213
25th Generation	0.4219±0.0754	0.1385±0.0463	0.0417±0.0271	0.3779±0.0550	0.0200±0.0235
50th Generation	0.5063±0.0723	0.1080±0.0327	0.0481±0.0332	0.3218±0.0516	0.0158±0.0161
100th Generation	0.5756±0.0759	0.0872±0.0223	0.0464±0.0393	0.2738±0.0516	0.0171±0.0129
VOW					
15th Generation	0.3472±0.0622	0.2032±0.0474	0.0249±0.0124	0.3900±0.0478	0.0348±0.0197
25th Generation	0.3563±0.0896	0.1657±0.0410	0.0210±0.0156	0.4265±0.0636	0.0305±0.0167
50th Generation	0.4166±0.1124	0.1369±0.0374	0.0185±0.0180	0.4049±0.0814	0.0231±0.0166
100th Generation	0.4975±0.1073	0.1078±0.0276	0.0207±0.0199	0.3446±0.0806	0.0294±0.0222
WAV					
15th Generation	0.5475±0.0624	0.1978±0.0431	0.0163±0.0176	0.2182±0.0354	0.0203±0.0116
25th Generation	0.5675±0.0618	0.1741±0.0453	0.0113±0.0187	0.2366±0.0333	0.0106±0.0091
50th Generation	0.6075±0.0468	0.1366±0.0322	0.0078±0.0123	0.2395±0.0293	0.0087±0.0065
100th Generation	0.6702±0.0348	0.1104±0.0226	0.0072±0.0082	0.2032±0.0221	0.0091±0.0062
YST					
15th Generation	0.1533±0.0704	0.1305±0.0435	0.0527±0.0401	0.6403±0.0890	0.0229±0.0132
25th Generation	0.1722±0.0741	0.1303±0.0599	0.0528±0.0583	0.6157±0.0912	0.0287±0.0205
50th Generation	0.2182±0.1029	0.1258±0.0453	0.0828±0.0643	0.5301±0.0736	0.0428±0.0310
100th Generation	0.2850±0.1140	0.1105±0.0300	0.0979±0.0722	0.4370±0.0682	0.0694±0.0553

Table 5.12: Evolution of the probability of selection of the original genetic operators on all eight datasets using the Euclidean Accuracy fitness function (EA-FF), and its standard deviation

	St-XO	Swap-dim	St-Mut	Add-dim	Rem-dim
HRT					
15th Generation	0.4065±0.1586	0.2542±0.0617	0.0724±0.0510	0.1782±0.1250	0.0887±0.0466
25th Generation	0.4145±0.1707	0.2349±0.0699	0.0722±0.0687	0.2178±0.1119	0.0605±0.0399
50th Generation	0.4527±0.1746	0.1869±0.0549	0.0920±0.0851	0.1984±0.0946	0.0700±0.0462
100th Generation	0.5005±0.1673	0.1499±0.0329	0.1014±0.0923	0.1674±0.0772	0.0808±0.0582
IM-3					
15th Generation	0.5445±0.1459	0.2261±0.0600	0.0523±0.0219	0.0779±0.1123	0.0991±0.0310
25th Generation	0.5817±0.1881	0.2030±0.0797	0.0434±0.0316	0.0986±0.1423	0.0734±0.0363
50th Generation	0.6185±0.1659	0.1813±0.0557	0.0589±0.0603	0.0846±0.0923	0.0566±0.0284
100th Generation	0.6463±0.1514	0.1572±0.0377	0.0607±0.0627	0.0783±0.0830	0.0575±0.0340
IM-10					
15th Generation	0.2924±0.0909	0.1659±0.0460	0.0596±0.0359	0.4502±0.0875	0.0319±0.0180
25th Generation	0.3656±0.0767	0.1353±0.0445	0.0588±0.0417	0.4132±0.0763	0.0271±0.0209
50th Generation	0.4560±0.0816	0.1143±0.0401	0.0450±0.0376	0.3606±0.0684	0.0241±0.0264
100th Generation	0.5316±0.0774	0.0978±0.0259	0.0604±0.0524	0.2875±0.0561	0.0227±0.0267
M-L					
15th Generation	0.3583±0.0786	0.1642±0.0367	0.0414±0.0251	0.4129±0.0624	0.0232±0.0137
25th Generation	0.4009±0.0880	0.1472±0.0425	0.0315±0.0255	0.4020±0.0791	0.0185±0.0122
50th Generation	0.5007±0.0827	0.1115±0.0351	0.0194±0.0168	0.3502±0.0655	0.0181±0.0105
100th Generation	0.6034±0.0659	0.0849±0.0228	0.0177±0.0169	0.2779±0.0544	0.0161±0.0125
SEG					
15th Generation	0.3418±0.0714	0.1956±0.0457	0.0675±0.0530	0.3633±0.0638	0.0318±0.0208
25th Generation	0.3937±0.1089	0.1493±0.0430	0.0617±0.0730	0.3717±0.0740	0.0236±0.0186
50th Generation	0.4936±0.0961	0.1096±0.0317	0.0499±0.0565	0.3263±0.0775	0.0206±0.0175
100th Generation	0.5730±0.0793	0.0876±0.0189	0.0561±0.0575	0.2629±0.0643	0.0204±0.0180
VOW					
15th Generation	0.3641±0.0715	0.1827±0.0406	0.0221±0.0116	0.3960±0.0669	0.0350±0.0221
25th Generation	0.3551±0.0993	0.1664±0.0393	0.0208±0.0201	0.4302±0.0655	0.0275±0.0235
50th Generation	0.4185±0.0911	0.1408±0.0303	0.0226±0.0216	0.3915±0.0625	0.0266±0.0238
100th Generation	0.5147±0.0724	0.1038±0.0254	0.0255±0.0248	0.3289±0.0529	0.0271±0.0229
WAV					
15th Generation	0.5170±0.0693	0.1954±0.0497	0.0186±0.0190	0.2384±0.0394	0.0305±0.0259
25th Generation	0.5558±0.0686	0.1594±0.0435	0.0142±0.0157	0.2531±0.0336	0.0176±0.0162
50th Generation	0.6004±0.0569	0.1259±0.0364	0.0098±0.0124	0.2502±0.0346	0.0137±0.0099
100th Generation	0.6664±0.0409	0.1041±0.0241	0.0081±0.0080	0.2112±0.0256	0.0103±0.0070
YST					
15th Generation	0.1722±0.0766	0.1365±0.0548	0.0376±0.0191	0.6318±0.0763	0.0219±0.0156
25th Generation	0.2143±0.1110	0.1185±0.0434	0.0366±0.0344	0.6066±0.1014	0.0240±0.0171
50th Generation	0.2722±0.1187	0.1039±0.0369	0.0491±0.0510	0.5379±0.1116	0.0368±0.0275
100th Generation	0.3687±0.1265	0.0998±0.0273	0.0584±0.0562	0.4295±0.1030	0.0436±0.0414

Table 5.13: Evolution of the probability of selection of the original genetic operators on all eight datasets using the first distance-based fitness function (ED1-FF), and its standard deviation

Using ten genetic operators

Here, we implemented five new genetic operators and ran the same experiment as the previous one, but using ten genetic operators. Having now ten operators, the initial probability of each individual is 10%. Although the standard deviation ratio to the probabilities is slightly higher than the one obtained when using five genetic operators, we think that we still have some ground to analyze the obtained probabilities.

Crossover 1 (St-XO): Like what was observed when using five genetic operators, this GO has an above average probability on the populations of all datasets. With the exception of the YST population, all population has the probability of this operator as at least 30%. This probability increases in all cases with the passing of the generations, once again indicating that as the population is growing, the need to restructure the individuals, using parts of the structure of others, increases. It seems to be more beneficial to use parts of the structure of other individuals to improve the population rather than using the mutation operators. It can be argued that this happens because the structure of others individuals was already partially adapted to the problem, while the mutation operators give a randomly generated branch.

Crossover 2 (Swap-dim): Although at the last generation, the probability of selecting this operator is still above average, the probability of selecting this operator is always decreasing over the generations. The only exception to this is the YST population that have a stable, average probability over the generations. This may indicate that after the individuals have dimensions that have evolved into something usable for classification, they won't benefit from swapping them with other individuals.

Mutation 1 (St-Mut): With both fitness functions, all the populations seemed to reject this operator. This conclusion is taken from the probability of selection being below 5% in all populations, in all generations. Since the **Add-dim** GO has a higher probability in most datasets, it might be safe to assume that the population will rather add a new dimension than grafting a new branch on the individual.

Mutation 2 (Add-dim): With the exception of the HRT, IM-3, and WAV populations, that give a stable probability to this GO of, respectively, 10%, 3%, and 19%, all other population give probabilities of selection between 28% and 47% on the 15^{th} generation, that decrease to a probability between 21% and 35%. This decrease is normal since as the individuals grow larger on the number of dimensions, they tend to stop benefiting from further increasing the number of dimensions.

Mutation 3 (Rem-dim): Like it was mentioned in the results from using five genetic operators, this GO's low probabilities of selection may result from the populations being able to learn when they should increase the number of dimensions, indicating that may not exist a need to have an operator that removes dimensions.

Mutation 4 to 7 (Grow or Trim, Trim, Chop, Singularitree): Like what was planned, these mutation operators are rejected by the classifier, having every one of them,

with both fitness functions, below average probabilities. The mutation operators **Grow or Trim** and **Trim** have a higher selection probability than the mutation operator **St-Mut**. This may be justified by them being similar to the original mutation operator and the population favoring these new methods over the original. We can allegedly say that this is caused by the Trim effect of both these genetic operators. It is possible that after a few generations, the individuals are able to benefit from random cuts in their structure.

Crossover 3 (Swap3-dim): This new crossover method is similar to **Swap-dim** with the exception that it works with three individuals instead of one, having a similar behavior on the evolution of the probabilities. With the exception of the YST populations that have a stable selection probability of 13%, all other populations begin with a selection probability between 17% and 24% on the 15^{th} generation and have a selection probability between 11% and 17% in the last generation. This may suggest that a crossover with many individuals may be preferable to the standard crossover.

Like what happened when we were using five genetic operators, the evolution of the probability over the generation tended to be similar between the EA populations and the ED1 population. Since this happened, the following plot graphs have all been made from results obtained from the EA populations.

Once again, each GO probability evolution showed similar behaviors when compared with the same GO in population learning other datasets. This can be seen in Figure 5.8. An interesting thing that was observed when using ten genetic operators was that the evolution of the probability of the GOs was not always stable. We could have already seen this in Figure 5.8 and this is even more clear in Figure 5.9. These two GO do not seem to have a clear line that allows us to predict the probability in later generations.

Lastly, we had concluded that the new crossover GO (**Swap3-dim**) had a better probability of selection than the original crossover GO that swapped dimensions (**Swap-dim**). This difference in the probability of selection of these two individuals is clear in Figure 5.10. For this plot graph, we selected the HRT and MCD-3 datasets as all datasets had a similar evolution, and the populations learning these two datasets gave them the highest probability of selection.

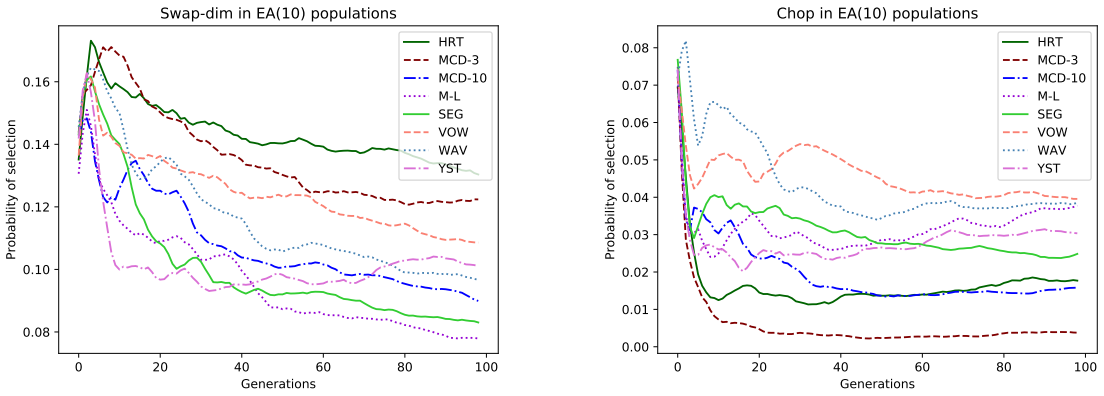

Figure 5.8: Evolution of probability of selection of the **Swap-dim**(left) and the **Chop**(right) GOs over the generations in all dataset, using the EA-FF and the ten GOs.

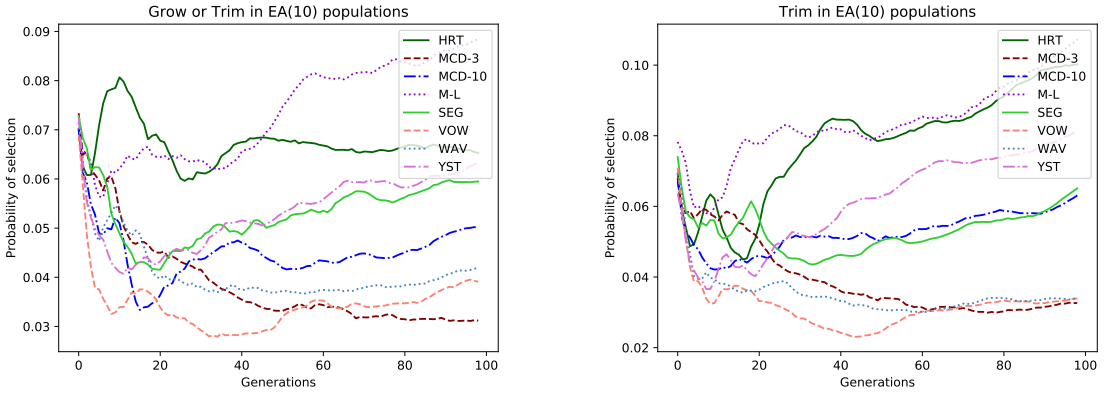

Figure 5.9: Evolution of probability of selection of the **Grow or Trim**(left) and the **Trim**(right) GOs over the generations in all dataset, using the EA-FF and the ten GOs.

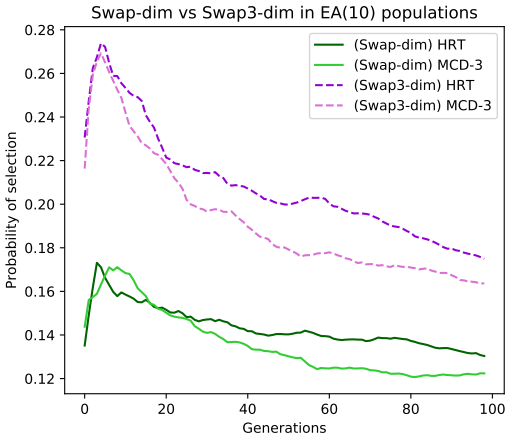

Figure 5.10: Evolution of probability of selection of the **Swap-dim**(solid line) and the **Swap3-dim**(dashed line) GOs over the generations in the HRT and MCD-3 dataset, using the EA-FF and the ten GOs.

	St-XO	Swap-dim	St-Mut	Add-dim	Rem-dim
HRT					
15th Generation	0.2814±0.1065	0.1549±0.0311	0.0231±0.0105	0.0971±0.0840	0.0491±0.0312
25th Generation	0.2964±0.1735	0.1518±0.0473	0.0263±0.0242	0.1211±0.0852	0.0393±0.0375
50th Generation	0.3308±0.1976	0.1408±0.0369	0.0219±0.0209	0.1116±0.0828	0.0283±0.0248
100th Generation	0.3489±0.1795	0.1305±0.0274	0.0240±0.0224	0.0948±0.0637	0.0389±0.0298
IM-3					
15th Generation	0.3406±0.0980	0.1617±0.0366	0.0445±0.0263	0.0310±0.0388	0.0751±0.0255
25th Generation	0.4147±0.1514	0.1500±0.0491	0.0356±0.0390	0.0327±0.0502	0.0591±0.0319
50th Generation	0.5123±0.1683	0.1329±0.0428	0.0243±0.0406	0.0307±0.0527	0.0453±0.0304
100th Generation	0.5504±0.1443	0.1231±0.0319	0.0203±0.0331	0.0290±0.0437	0.0420±0.0276
IM-10					
15th Generation	0.1627±0.0812	0.1341±0.0481	0.0374±0.0253	0.3339±0.0631	0.0203±0.0141
25th Generation	0.2048±0.0932	0.1249±0.0372	0.0386±0.0478	0.3342±0.0700	0.0156±0.0149
50th Generation	0.2970±0.1081	0.1012±0.0325	0.0435±0.0586	0.2882±0.0570	0.0160±0.0156
100th Generation	0.3692±0.1034	0.0904±0.0285	0.0410±0.0487	0.2425±0.0449	0.0160±0.0164
M-L					
15th Generation	0.2209±0.0711	0.1118±0.0339	0.0223±0.0188	0.3035±0.0590	0.0130±0.0096
25th Generation	0.2338±0.0882	0.1102±0.0423	0.0170±0.0190	0.2960±0.0698	0.0109±0.0130
50th Generation	0.2977±0.1151	0.0879±0.0370	0.0106±0.0143	0.2742±0.0558	0.0110±0.0119
100th Generation	0.3334±0.1021	0.0783±0.0261	0.0110±0.0116	0.2118±0.0448	0.0103±0.0129
SEG					
15th Generation	0.1980±0.0858	0.1267±0.0448	0.0365±0.0345	0.2854±0.0688	0.0181±0.0146
25th Generation	0.2421±0.1206	0.1025±0.0483	0.0344±0.0388	0.2933±0.0753	0.0146±0.0141
50th Generation	0.3192±0.1138	0.0921±0.0395	0.0343±0.0353	0.2701±0.0730	0.0116±0.0104
100th Generation	0.3860±0.0992	0.0833±0.0292	0.0318±0.0333	0.2240±0.0583	0.0132±0.0129
VOW					
15th Generation	0.2007±0.0670	0.1364±0.0435	0.0159±0.0199	0.3081±0.0385	0.0147±0.0051
25th Generation	0.1813±0.0699	0.1325±0.0400	0.0157±0.0284	0.3599±0.0500	0.0116±0.0082
50th Generation	0.2198±0.0854	0.1244±0.0393	0.0166±0.0216	0.3471±0.0525	0.0189±0.0172
100th Generation	0.2996±0.0964	0.1097±0.0298	0.0233±0.0268	0.2846±0.0487	0.0224±0.0177
WAV					
15th Generation	0.2900±0.0607	0.1339±0.0333	0.0131±0.0102	0.1949±0.0302	0.0121±0.0103
25th Generation	0.3394±0.0985	0.1348±0.0381	0.0090±0.0132	0.2193±0.0359	0.0087±0.0085
50th Generation	0.4015±0.0780	0.1063±0.0272	0.0055±0.0080	0.2266±0.0350	0.0070±0.0078
100th Generation	0.4352±0.0725	0.0969±0.0213	0.0046±0.0059	0.1965±0.0260	0.0077±0.0062
YST					
15th Generation	0.1023±0.0550	0.1008±0.0394	0.0420±0.0339	0.4754±0.0650	0.0137±0.0145
25th Generation	0.0929±0.0519	0.0986±0.0330	0.0399±0.0608	0.4862±0.0793	0.0184±0.0214
50th Generation	0.1361±0.0771	0.0986±0.0299	0.0262±0.0367	0.4384±0.0875	0.0195±0.0152
100th Generation	0.1681±0.0940	0.1016±0.0317	0.0251±0.0289	0.3585±0.0721	0.0337±0.0328

Table 5.14: Evolution of the probability of selection of the original five genetic operators when all ten genetic operators were tested, and their standard deviation. Results obtained from using the EA-FF

	Grow or Trim	Trim	Chop	Singularitree	Swap3-dim
HRT					
15th Generation	0.0765±0.0434	0.0485±0.0250	0.0146±0.0245	0.0054±0.0030	0.2494±0.0549
25th Generation	0.0641±0.0490	0.0654±0.0687	0.0141±0.0230	0.0022±0.0008	0.2192±0.0605
50th Generation	0.0697±0.0544	0.0795±0.0818	0.0139±0.0306	0.0012±0.0014	0.2022±0.0571
100th Generation	0.0665±0.0375	0.1027±0.0988	0.0179±0.0253	0.0007±0.0015	0.1752±0.0402
IM-3					
15th Generation	0.0472±0.0195	0.0580±0.0302	0.0063±0.0040	0.0042±0.0018	0.2314±0.0433
25th Generation	0.0451±0.0308	0.0458±0.0325	0.0038±0.0031	0.0020±0.0006	0.2112±0.0441
50th Generation	0.0338±0.0256	0.0354±0.0412	0.0024±0.0025	0.0010±0.0004	0.1818±0.0507
100th Generation	0.0315±0.0235	0.0349±0.0480	0.0039±0.0070	0.0007±0.0005	0.1643±0.0366
IM-10					
15th Generation	0.0387±0.0286	0.0433±0.0254	0.0337±0.0279	0.0046±0.0039	0.1913±0.0520
25th Generation	0.0392±0.0241	0.0463±0.0285	0.0243±0.0253	0.0018±0.0004	0.1704±0.0400
50th Generation	0.0431±0.0320	0.0519±0.0455	0.0140±0.0155	0.0006±0.0002	0.1446±0.0370
100th Generation	0.0501±0.0332	0.0633±0.0446	0.0157±0.0147	0.0002±0.0001	0.1115±0.0287
M-L					
15th Generation	0.0642±0.0386	0.0705±0.0278	0.0301±0.0285	0.0035±0.0012	0.1602±0.0444
25th Generation	0.0646±0.0411	0.0806±0.0567	0.0296±0.0349	0.0024±0.0049	0.1549±0.0406
50th Generation	0.0715±0.0537	0.0798±0.0613	0.0277±0.0409	0.0013±0.0041	0.1383±0.0368
100th Generation	0.0882±0.0591	0.1085±0.0770	0.0375±0.0416	0.0006±0.0020	0.1203±0.0325
SEG					
15th Generation	0.0448±0.0255	0.0518±0.0241	0.0372±0.0362	0.0047±0.0046	0.1968±0.0537
25th Generation	0.0453±0.0376	0.0496±0.0322	0.0377±0.0399	0.0021±0.0012	0.1785±0.0442
50th Generation	0.0511±0.0395	0.0491±0.0351	0.0283±0.0314	0.0012±0.0030	0.1430±0.0448
100th Generation	0.0597±0.0388	0.0649±0.0420	0.0249±0.0233	0.0006±0.0021	0.1116±0.0283
VOW					
15th Generation	0.0371±0.0298	0.0369±0.0324	0.0511±0.0409	0.0064±0.0074	0.1928±0.0397
25th Generation	0.0325±0.0241	0.0324±0.0283	0.0475±0.0362	0.0027±0.0032	0.1839±0.0456
50th Generation	0.0296±0.0234	0.0239±0.0259	0.0457±0.0471	0.0007±0.0002	0.1734±0.0433
100th Generation	0.0392±0.0326	0.0338±0.0301	0.0393±0.0335	0.0002±0.0001	0.1479±0.0317
WAV					
15th Generation	0.0484±0.0380	0.0376±0.0152	0.0618±0.0635	0.0125±0.0186	0.1957±0.0440
25th Generation	0.0392±0.0456	0.0381±0.0246	0.0487±0.0450	0.0059±0.0132	0.1569±0.0524
50th Generation	0.0367±0.0390	0.0310±0.0246	0.0343±0.0343	0.0022±0.0059	0.1488±0.0423
100th Generation	0.0422±0.0378	0.0342±0.0268	0.0384±0.0315	0.0010±0.0029	0.1432±0.0346
YST					
15th Generation	0.0416±0.0252	0.0447±0.0313	0.0238±0.0177	0.0036±0.0010	0.1522±0.0445
25th Generation	0.0452±0.0531	0.0472±0.0413	0.0250±0.0354	0.0017±0.0004	0.1448±0.0636
50th Generation	0.0516±0.0545	0.0621±0.0570	0.0265±0.0320	0.0008±0.0003	0.1403±0.0426
100th Generation	0.0630±0.0496	0.0809±0.0664	0.0304±0.0356	0.0003±0.0001	0.1385±0.0341

Table 5.15: Evolution of the probability of selection of the new five genetic operators when all ten genetic operators were tested, and their standard deviation. Results obtained from using the EA-FF

	St-XO	Swap-dim	St-Mut	Add-dim	Rem-dim
HRT					
15th Generation	0.2868±0.1073	0.1601±0.0445	0.0408±0.0311	0.0789±0.0642	0.0525±0.0302
25th Generation	0.2975±0.1591	0.1529±0.0509	0.0381±0.0450	0.1062±0.0849	0.0407±0.0376
50th Generation	0.3559±0.1915	0.1355±0.0453	0.0260±0.0384	0.0928±0.0834	0.0287±0.0219
100th Generation	0.3848±0.1655	0.1287±0.0335	0.0265±0.0288	0.0866±0.0690	0.0297±0.0200
IM-3					
15th Generation	0.3234±0.1105	0.1613±0.0373	0.0401±0.0421	0.0532±0.0637	0.0627±0.0289
25th Generation	0.3906±0.1628	0.1514±0.0388	0.0320±0.0328	0.0623±0.0834	0.0458±0.0297
50th Generation	0.4717±0.1922	0.1322±0.0411	0.0194±0.0211	0.0584±0.0802	0.0405±0.0267
100th Generation	0.5087±0.1771	0.1237±0.0306	0.0225±0.0253	0.0469±0.0604	0.0389±0.0271
IM-10					
15th Generation	0.1635±0.0704	0.1240±0.0386	0.0391±0.0289	0.3351±0.0867	0.0206±0.0133
25th Generation	0.2162±0.0831	0.1134±0.0385	0.0433±0.0363	0.3146±0.0688	0.0165±0.0142
50th Generation	0.2609±0.1026	0.0995±0.0404	0.0499±0.0389	0.2852±0.0753	0.0172±0.0188
100th Generation	0.3390±0.0967	0.0877±0.0312	0.0536±0.0407	0.2431±0.0721	0.0212±0.0198
M-L					
15th Generation	0.2290±0.0932	0.1071±0.0369	0.0188±0.0131	0.3214±0.0587	0.0107±0.0047
25th Generation	0.2810±0.1059	0.0978±0.0451	0.0144±0.0140	0.3079±0.0552	0.0073±0.0044
50th Generation	0.3134±0.1104	0.0977±0.0355	0.0126±0.0154	0.2728±0.0587	0.0073±0.0098
100th Generation	0.3578±0.1177	0.0834±0.0210	0.0102±0.0122	0.2193±0.0448	0.0098±0.0108
SEG					
15th Generation	0.2090±0.0972	0.1364±0.0408	0.0227±0.0097	0.2615±0.0617	0.0141±0.0105
25th Generation	0.2664±0.0917	0.1199±0.0494	0.0246±0.0290	0.2784±0.0735	0.0099±0.0094
50th Generation	0.3299±0.1039	0.0856±0.0310	0.0178±0.0164	0.2539±0.0633	0.0130±0.0152
100th Generation	0.3888±0.0922	0.0801±0.0254	0.0199±0.0195	0.2147±0.0474	0.0133±0.0148
VOW					
15th Generation	0.2100±0.0801	0.1308±0.0317	0.0166±0.0149	0.3120±0.0514	0.0132±0.0058
25th Generation	0.2094±0.0973	0.1179±0.0404	0.0157±0.0168	0.3580±0.0540	0.0119±0.0064
50th Generation	0.2324±0.1044	0.1133±0.0271	0.0178±0.0174	0.3400±0.0658	0.0127±0.0109
100th Generation	0.3167±0.1000	0.0975±0.0245	0.0197±0.0199	0.3013±0.0562	0.0232±0.0209
WAV					
15th Generation	0.3044±0.0842	0.1255±0.0372	0.0061±0.0033	0.1974±0.0437	0.0125±0.0109
25th Generation	0.3177±0.0939	0.1291±0.0438	0.0052±0.0083	0.2147±0.0395	0.0081±0.0102
50th Generation	0.3669±0.0897	0.1078±0.0284	0.0034±0.0057	0.2248±0.0289	0.0068±0.0061
100th Generation	0.4247±0.0686	0.0937±0.0177	0.0026±0.0039	0.1943±0.0207	0.0068±0.0052
YST					
15th Generation	0.1288±0.0587	0.0950±0.0337	0.0279±0.0221	0.4821±0.0522	0.0135±0.0091
25th Generation	0.1383±0.0888	0.1063±0.0428	0.0369±0.0253	0.4520±0.0727	0.0106±0.0140
50th Generation	0.1749±0.1021	0.1084±0.0392	0.0397±0.0345	0.3861±0.0623	0.0162±0.0160
100th Generation	0.2126±0.1126	0.1022±0.0315	0.0456±0.0313	0.3323±0.0625	0.0294±0.0336

Table 5.16: Evolution of the probability of selection of the original five genetic operators when all ten genetic operators were tested, and their standard deviation. Results obtained from using the ED1-FF

	Grow or Trim	Trim	Chop	Singularitree	Swap3-dim
HRT					
15th Generation	0.0642±0.0368	0.0587±0.0397	0.0098±0.0070	0.0056±0.0023	0.2426±0.0455
25th Generation	0.0715±0.0637	0.0575±0.0403	0.0141±0.0201	0.0024±0.0012	0.2191±0.0626
50th Generation	0.0632±0.0492	0.0784±0.0729	0.0135±0.0171	0.0018±0.0041	0.2043±0.0536
100th Generation	0.0623±0.0454	0.0861±0.0694	0.0172±0.0182	0.0008±0.0020	0.1772±0.0285
IM-3					
15th Generation	0.0554±0.0319	0.0672±0.0487	0.0104±0.0166	0.0049±0.0024	0.2215±0.0511
25th Generation	0.0463±0.0360	0.0589±0.0380	0.0135±0.0254	0.0029±0.0040	0.1963±0.0572
50th Generation	0.0331±0.0286	0.0528±0.0477	0.0131±0.0242	0.0019±0.0053	0.1767±0.0501
100th Generation	0.0337±0.0282	0.0499±0.0512	0.0159±0.0264	0.0005±0.0003	0.1593±0.0343
IM-10					
15th Generation	0.0472±0.0293	0.0448±0.0232	0.0278±0.0134	0.0059±0.0118	0.1920±0.0576
25th Generation	0.0499±0.0425	0.0486±0.0328	0.0203±0.0238	0.0026±0.0046	0.1745±0.0472
50th Generation	0.0644±0.0473	0.0580±0.0389	0.0135±0.0150	0.0013±0.0039	0.1502±0.0358
100th Generation	0.0579±0.0418	0.0638±0.0486	0.0148±0.0128	0.0006±0.0020	0.1183±0.0292
M-L					
15th Generation	0.0498±0.0365	0.0668±0.0428	0.0223±0.0222	0.0037±0.0008	0.1704±0.0436
25th Generation	0.0524±0.0344	0.0651±0.0416	0.0186±0.0266	0.0016±0.0004	0.1539±0.0493
50th Generation	0.0576±0.0421	0.0837±0.0668	0.0197±0.0212	0.0007±0.0002	0.1347±0.0400
100th Generation	0.0621±0.0564	0.1274±0.0906	0.0198±0.0194	0.0002±0.0001	0.1100±0.0302
SEG					
15th Generation	0.0568±0.0316	0.0562±0.0331	0.0387±0.0344	0.0048±0.0032	0.1998±0.0515
25th Generation	0.0532±0.0386	0.0498±0.0373	0.0348±0.0317	0.0016±0.0004	0.1613±0.0483
50th Generation	0.0703±0.0554	0.0518±0.0459	0.0341±0.0297	0.0006±0.0002	0.1430±0.0416
100th Generation	0.0714±0.0524	0.0607±0.0453	0.0292±0.0243	0.0002±0.0001	0.1215±0.0282
VOW					
15th Generation	0.0372±0.0214	0.0352±0.0200	0.0404±0.0227	0.0060±0.0059	0.1985±0.0491
25th Generation	0.0373±0.0238	0.0258±0.0211	0.0302±0.0261	0.0042±0.0087	0.1896±0.0559
50th Generation	0.0500±0.0440	0.0328±0.0332	0.0325±0.0268	0.0015±0.0045	0.1670±0.0450
100th Generation	0.0442±0.0299	0.0368±0.0316	0.0296±0.0258	0.0006±0.0020	0.1304±0.0275
WAV					
15th Generation	0.0398±0.0331	0.0403±0.0275	0.0588±0.0364	0.0124±0.0159	0.2028±0.0450
25th Generation	0.0332±0.0366	0.0402±0.0440	0.0600±0.0535	0.0049±0.0099	0.1869±0.0542
50th Generation	0.0357±0.0377	0.0403±0.0473	0.0509±0.0366	0.0021±0.0056	0.1613±0.0424
100th Generation	0.0352±0.0390	0.0406±0.0410	0.0591±0.0509	0.0010±0.0030	0.1420±0.0238
YST					
15th Generation	0.0486±0.0374	0.0378±0.0237	0.0267±0.0169	0.0043±0.0053	0.1352±0.0367
25th Generation	0.0508±0.0493	0.0434±0.0332	0.0229±0.0224	0.0017±0.0004	0.1371±0.0405
50th Generation	0.0471±0.0496	0.0523±0.0427	0.0244±0.0285	0.0007±0.0002	0.1501±0.0506
100th Generation	0.0542±0.0413	0.0665±0.0445	0.0252±0.0210	0.0003±0.0001	0.1316±0.0332

Table 5.17: Evolution of the probability of selection of the new five genetic operators when all ten genetic operators were tested, and their standard deviation. Results obtained from using the ED1-FF

Overall conclusions

These results allowed us to take a few conclusions such as:

- The crossover operators that swap dimensions between individuals, are more useful on early stages of the evolution;
- The increase of the number of dimensions on early stages is related to the number of classes but even if the dataset has many classes, the population can learn to quickly stop increasing the number of dimensions if it doesn't improve the individual;
- Although the **St-XO** GO is always useful, it is even more preferred in the later stages of the evolution;
- Although the new operators did not increase the accuracy of the populations, they reduced the size of the individuals. This suggests that it might be worth to develop new genetic operators as the **St-XO** and **St-Mut** were less preferred than the new GOs that were developed having them as as inspiration (**Grow or Trim**, **Trim**, and **Swap3-dim**);

Chapter 6

Conclusions

6.1 Results

This chapter contains the conclusions about all the three stages of the project, commentaries about our approaches and problems found, and some future work we have planned.

The first stage of the project was the implementation of our M3GP classifier, and a comparison of the results obtained when using the same parameters and datasets as those in [4]. The individuals from the populations created by our implementation have a greater number of dimensions and have a larger size than those from the original implementation of the M3GP. This difference in the number of dimensions and the size of the individuals may have been due to the original M3GP being implemented over the GPLAB toolbox [39] which contains additional bloat control [37, 38] measures set by default. Although we had this problem, since the results were not necessarily worse than those of the original implementation, we decided to keep this version and continue to the next stages, knowing that improving the classifier could lead to better results on the following stages of this project.

The second stage was the implementation of two distance-based fitness functions as an attempt to replace the currently used accuracy-based fitness function. All fitness functions in this stage used the Euclidean distance rather than the Mahalanobis distance. This choice was made in order to have faster results. The results obtained from both distance-based fitness functions were compared with an accuracy-based fitness function. One of the distance-based fitness functions showed bad results and was dismissed. The other fitness function achieved results not significantly different from the accuracy-based fitness function in both training accuracy and test accuracy, with the exception of one problem where borderline significantly worse results were observed. This distance-based fitness function also maintained the number of dimensions, size of the individuals, and the overall evolution of the population while being a fitness function that has a lower computational complexity. During this stage, we had a problem with the creation of the fitness func-

tion. Since distance values tend to be greater in spaces in higher dimensions, we had to spend some time trying to find a way to normalize distance between dimensions, in order to compare individuals with different numbers of dimensions. The fitness function we selected gave results that are equivalent to using an accuracy-based fitness function, but we think it's possible to create a better fitness function, although it might not be trivial to find a good one.

The final stage was the implementation of a new method for the selection of genetic operators, that evolves the probabilities of each genetic operator to the needs of the population. Five new genetic operators were also created. The results obtained here were quite positive, showing that both the populations using accuracy-based and distance-based fitness functions had significantly better training results in four of the eight datasets, than their counter parts that use fixed probabilities of selection of the genetic operators. In both accuracy-based and distance-based populations, the test results were significantly better in one dataset and borderline better with one dataset, indicating that it is worth to explore an automatic adaptation of genetic operators' probabilities.

The results from the final stage also indicate that the standard crossover is always useful but it is even better on later stages of the evolution; that the dimension-swapping crossover tends to lose its value over the generations; that datasets with a higher number of classes tend to need a higher number of dimensions to be learned; and that crossover operators that use more individuals should be explored, as the one we implemented was prefered over the equivalent operator that used two individuals. Overall, this method of adaptation of probabilities of the genetic operators improved the training accuracy; allowed us to identify good and bad operators; gave us good information about a reasonable number of genetic operators by identifying those who may be worth to explore; and suggested that is might be worth to explore new genetic operators that are not included in standard GP.

Unfortunatly, although we used a distance-based fitness function as an attempt to improve the populations and used a method for automatic adaptation of genetic operator probabilities, we could not obtain the same results as those obtained when using the Mahalanobis distance.

6.2 Future Work

Over the course of this project, we had a few ideas of future work, some related with issues with our approach with the M3GP algorithm, some with issues with the GP algorithm in general, and some were just ideas we took from this project. Some of the future work we have planned is:

- We intend to make an implementation of distance-based fitness functions on the M2GP. Since all individuals on the M2GP have the same number of dimensions,

the results from this experiment are hoped to have better results than those obtained with the M3GP.

- We intend to make an implementation of a Geometric Semantic approach for multiclass classification. An issue with GP algorithms is that they tend to be slow when compared to other branches of machine learning. This was the only reason why we used the Euclidean distance on this project rather than the Mahalanobis distance, to avoid making even slower generations. After experimenting the GSGP approach, described in *An Introduction to Geometric Semantic Genetic Programming* (2017) [35], one of our goals was to make an implementation that used Geometric Semantic on the M2GP algorithm.

- Lastly, since the M3GP implicitly makes feature evolution, one of our goals, that is already in progress, is to attempt to use the M3GP algorithm as a pre-processor of datasets to be used together with other machine learning algorithms.

Bibliography

[1] Riccardo Poli and William B. Langdon and Nicholas Freitag McPhee. A field guide to genetic programming. Published via http://lulu.com and freely available at http://www.gp-field-guide.org.uk (2008). With contforensicriFs by J.R. Koza.

[2] Multiclass Classification Through Multidimensional Clustering. S Silva, L Munoz, L Trujillo, V Ingalalli, M Castelli, L Vanneschi. Genetic Programming Theory and Practice XIII (2016), 219-239.

[3] J. R. Koza. Genetic Programming: vol. 1, On the programming of computers by means of natural selection, volume 1. MIT press, 1992.

[4] Muñoz L., Silva S., Trujillo L. (2015) M3GP – Multiclass Classification with GP. In: Machado P. et al. (eds) Genetic Programming. EuroGP 2015. Lecture Notes in Computer Science, vol 9025. Springer, Cham

[5] Courrieu, Pierre. (2008). Fast Computation of Moore-Penrose Inverse Matrices. Neural Information Processing-Letters and Reviews. 8. .

[6] Eibe Frank, Mark A. Hall, and Ian H. Witten (2016). The WEKA Workbench. Online Appendix for "Data Mining: Practical Machine Learning Tools and Techniques", Morgan Kaufmann, Fourth Edition, 2016.

[7] J. R. Koza. Human-competitive results produced by genetic programming. Genetic Programming and Evolvable Machines, 11(3-4):251–284, Sept. 2010.

[8] V. Ingalalli, S. Silva, M. Castelli, and L. Vanneschi. A multi-dimensional genetic programming approach for multi-class classification problems. In M. Nicolau et al., editors, 17th European Conference on Genetic Programming, volume 8599 of LNCS, pages 48–60, Granada, Spain, 2014. Springer.

[9] K. Bache and M. Lichman. UCI machine learning repository, university of california, irvine, school of information and computer sciences. http://archive.ics.uci.edu/ml, 2017.

[10] U.S. geological survey (USGS) earth resources observation systems (EROS) data center (EDC). http://glovis.usgs.gov/.

[11] https://www.math.hmc.edu/funfacts/ffiles/20007.2.shtml

[12] Mahalanobis, Prasanta Chandra (1936). "On the generalised distance in statistics". Proceedings of the National Institute of Sciences of India. 2 (1): 49–55. Retrieved 2016-09-27.

[13] Frédéric Ratle, Christian Gagné, Anne-Laure Terrettaz-Zufferey, Mikhail Kanevski, Pierre Esseiva, Olivier Ribaux, Advanced clustering methods for mining chemical databases in forensic science, In Chemometrics and Intelligent Laboratory Systems, Volume 90, Issue 2, 2008, Pages 123-131, ISSN 0169-7439.

[14] J. P. Papa, L. P. Papa, D. R. Pereira and R. J. Pisani, "A Hyperheuristic Approach for Unsupervised Land-Cover Classification," in IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing, vol. 9, no. 6, pp. 2333-2342, June 2016.

[15] N. P. Shetty, "Using clustering to capture attackers," 2016 International Conference on Inventive Computation Technologies (ICICT), Coimbatore, 2016, pp. 1-5.

[16] Smart W., Zhang M. (2004) Probability Based Genetic Programming for Multiclass Object Classification. In: Zhang C., W. Guesgen H., Yeap WK. (eds) PRICAI 2004: Trends in Artificial Intelligence. PRICAI 2004. Lecture Notes in Computer Science, vol 3157. Springer, Berlin, Heidelberg

[17] Andrew Lensen, Bing Xue, and Mengjie Zhang. 2017. GPGC: genetic programming for automatic clustering using a flexible non-hyper-spherical graph-based approach. In Proceedings of the Genetic and Evolutionary Computation Conference (GECCO '17). ACM, New York, NY, USA, 449-456.

[18] Chih-Hung Wu, Hung-Ju Chou, and Wei-Han Su. 2008. Direct transformation of coordinates for GPS positioning using the techniques of genetic programming and symbolic regression. Eng. Appl. Artif. Intell. 21, 8 (December 2008), 1347-1359. DOI=http://dx.doi.org/10.1016/j.engappai.2008.02.001

[19] Ingalalli V., Silva S., Castelli M., Vanneschi L. (2014) A Multi-dimensional Genetic Programming Approach for Multi-class Classification Problems. In: Nicolau M. et al. (eds) Genetic Programming. EuroGP 2014. Lecture Notes in Computer Science, vol 8599. Springer, Berlin, Heidelberg

[20] La Cava, William & Silva, Sara & Danai, Kourosh & Spector, Lee & Vanneschi, Leonardo & Moore, Jason. (2018). Multidimensional genetic programming for multiclass classification. Swarm and Evolutionary Computation. 10.1016/j.swevo.2018.03.015.

[21] Ashish Kumar Patnaik, Prasanta Kumar Bhuyan, Application of genetic programming clustering in defining LOS criteria of urban street in Indian context, Travel Behaviour and Society, Volume 3, 2016, Pages 38-50, ISSN 2214-367X, https://doi.org/10.1016/j.tbs.2015.08.003.

[22] Downey C. & Zhang M. & Browne W. (2010), New crossover operators in linear genetic programming for multiclass object classification. In Proceedings of the 12th annual conference on Genetic and evolutionary computation (GECCO '10). ACM, New York, NY, USA, 885-892.

[23] N. Al-Madi and S. A. Ludwig, "Improving genetic programming classification for binary and multiclass datasets," 2013 IEEE Symposium on Computational Intelligence and Data Mining (CIDM), Singapore, 2013, pp. 166-173. doi: 10.1109/CIDM.2013.6597232

[24] Will Smart and Mengjie Zhang, "Using Genetic Programming for Multiclass Classification by Simultaneously Solving Component Binary Classification Problems", "Computer Science, Victoria University of Wellington", 2005, CS-TR-05-1, New Zealand

[25] Kun-Hong Liu and Chun-Gui Xu (2009), A genetic programming-based approach to the classification of multiclass microarray datasets, Bioinformatics, vol 25, pages 331-337

[26] Zhang M., Johnston M. (2009) A Variant Program Structure in Tree-Based Genetic Programming for Multiclass Object Classification. In: Cagnoni S. (eds) Evolutionary Image Analysis and Signal Processing. Studies in Computational Intelligence, vol 213. Springer, Berlin, Heidelberg

[27] Zhang, Y., & Rockett, P. I. (2009). A generic multi-dimensional feature extraction method using multiobjective genetic programming. Evolutionary Computation, 17(1), 89-115.

[28] Sherrah, J. R., Bogner, R. E., & Bouzerdoum, A. (1997). The evolutionary preprocessor: Automatic feature extraction for supervised classification using genetic programming. Genetic Programming, 304-312.

[29] Ling Guo, Daniel Rivero, Julián Dorado, Cristian R. Munteanu, Alejandro Pazos, Automatic feature extraction using genetic programming: An application to epileptic EEG classification, Expert Systems with Applications, Volume 38, Issue 8, 2011, Pages 10425-10436, ISSN 0957-4174,

[30] Tuson, Andrew. (1998). Adapting Operator Probabilities In Genetic Algorithms.

[31] Niehaus J., Banzhaf W. (2001) Adaption of Operator Probabilities in Genetic Programming. In: Miller J., Tomassini M., Lanzi P.L., Ryan C., Tettamanzi A.G.B., Langdon W.B. (eds) Genetic Programming. EuroGP 2001. Lecture Notes in Computer Science, vol 2038. Springer, Berlin, Heidelberg

[32] Darwin, Charles, 1809-1882. On The Origin of Species by Means of Natural Selection, or Preservation of Favoured Races in the Struggle for Life. London :John Murray, 1859.

[33] Sipper, Moshe et al. "Investigating the Parameter Space of Evolutionary Algorithms." BioData Mining 11 (2018): 2. PMC. Web. 28 June 2018.

[34] Arnold C. Evolution Runs Faster on Short Timescales. 2017. Quanta Magazine. www.quantamagazine.org/20170314-time-dependent-rate-phenomenon-evolution-viruses. Accessed 14 Mar 2017.

[35] Vanneschi L. (2017) An Introduction to Geometric Semantic Genetic Programming. In: Schütze O., Trujillo L., Legrand P., Maldonado Y. (eds) NEO 2015. Studies in Computational Intelligence, vol 663. Springer, Cham

[36] Pedro Domingos. 2012. A few useful things to know about machine learning. Commun. ACM 55, 10 (October 2012), 78-87.

[37] S. Silva & E. Costa (2009), Dynamic limits for bloat control in genetic programming and a review of past and current bloat theories, Genetic Programming and Evolvable Machines 10 (2), 141–179

[38] S. Silva & S. Dignum & L. Vanneschi (2012), Operator equalisation for bloat free genetic programming and a survey of bloat control methods, Genetic Programming and Evolvable Machines 13 (2), 197-238, 2012

[39] S. Silva & J. Almeida (2005), Gplab - a genetic programming toolbox for matlab, In Proc. of the Nordic MATLAB Conference (NMC-2003), 273–278

[40] Java(TM) SE Runtime Environment (build 1.8.0_151-b12), https://docs.oracle.com/javase/8/docs/api/

[41] R Core Team (2017). R: A language and environment for statistical computing. R Foundation for Statistical Computing, Vienna, Austria. https://www.R-project.org/.

[42] Hunter, J. D. (2007), Matplotlib: A 2D graphics environment, Computing In Science & Engineering, 9, 3, 90–95

[43] Python Software Foundation. Python Language Reference, version 3.6. Available at http://www.python.org

Glossary

eM3GP	ensemble M3GP
EA	Euclidean Accuracy
EC	Evolutionary Computation
ED1	Euclidean Distance 1
ED2	Euclidean Distance 2
Euc	Euclidean
FF	Fitness Function
GA	Genetic Algorithm
GO	Genetic Operator
GP	Genetic Programming
GPGC	Graph-based non-hyper-spherical Clustering GP
GPS	Global Positioning System
HRT	Heart
IM-3	Image 3
IM-10	Image 10
LGP	Linear Genetic Programming
M-L	Libras Movement
M2GP	Multidimentional Multiclass Genetic Programming
M3GP	M2GP with Multidimensional Populations
M4GP	M3GP with stack representation and lexicase parent selection
Mah	Mahalanobis
MLP	Multilayer Perceptron
PG	Programação Genética
SEG	Image Segmentation
VOW	Vowel
YST	Yeast
WAV	Waveform

Appendix A

Implementation

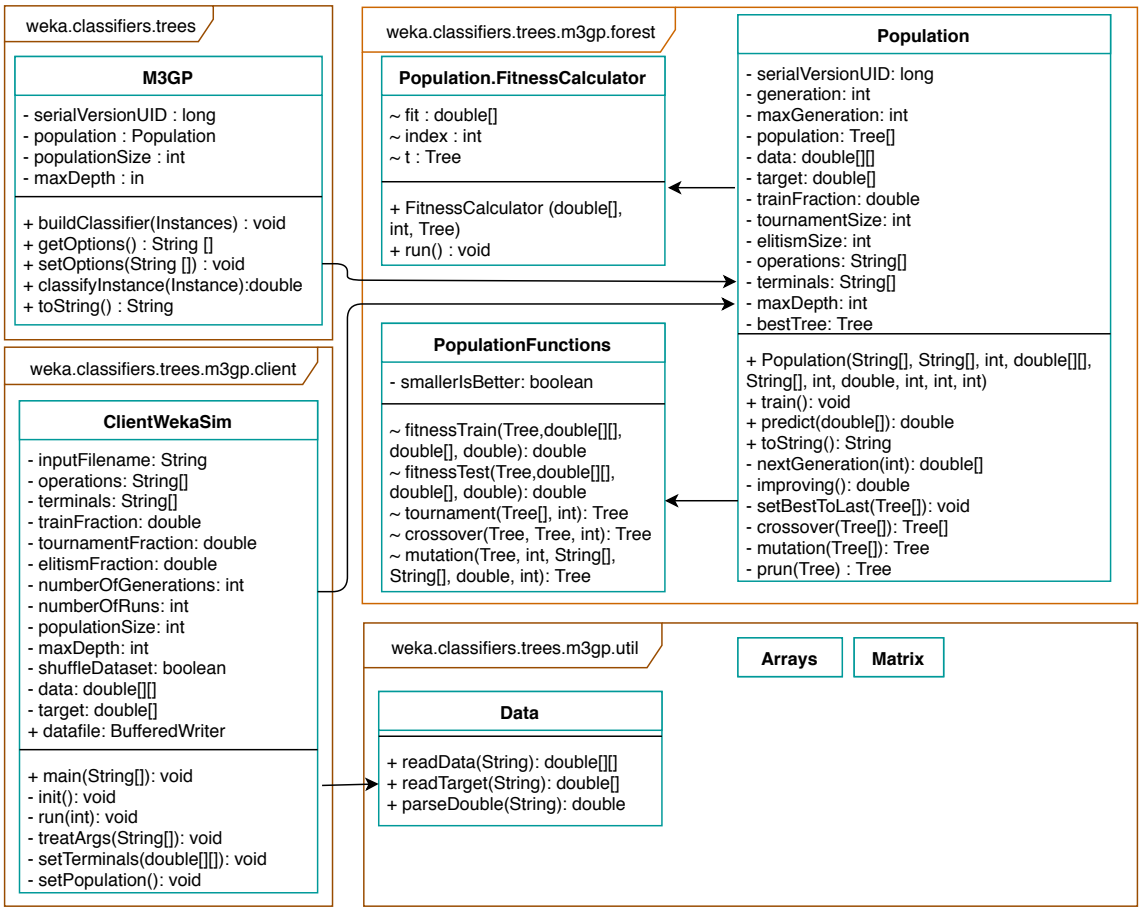

Figure A.1: Class Diagram section of the implementation used for the first stage of the project, referent to the initialization of the classifier

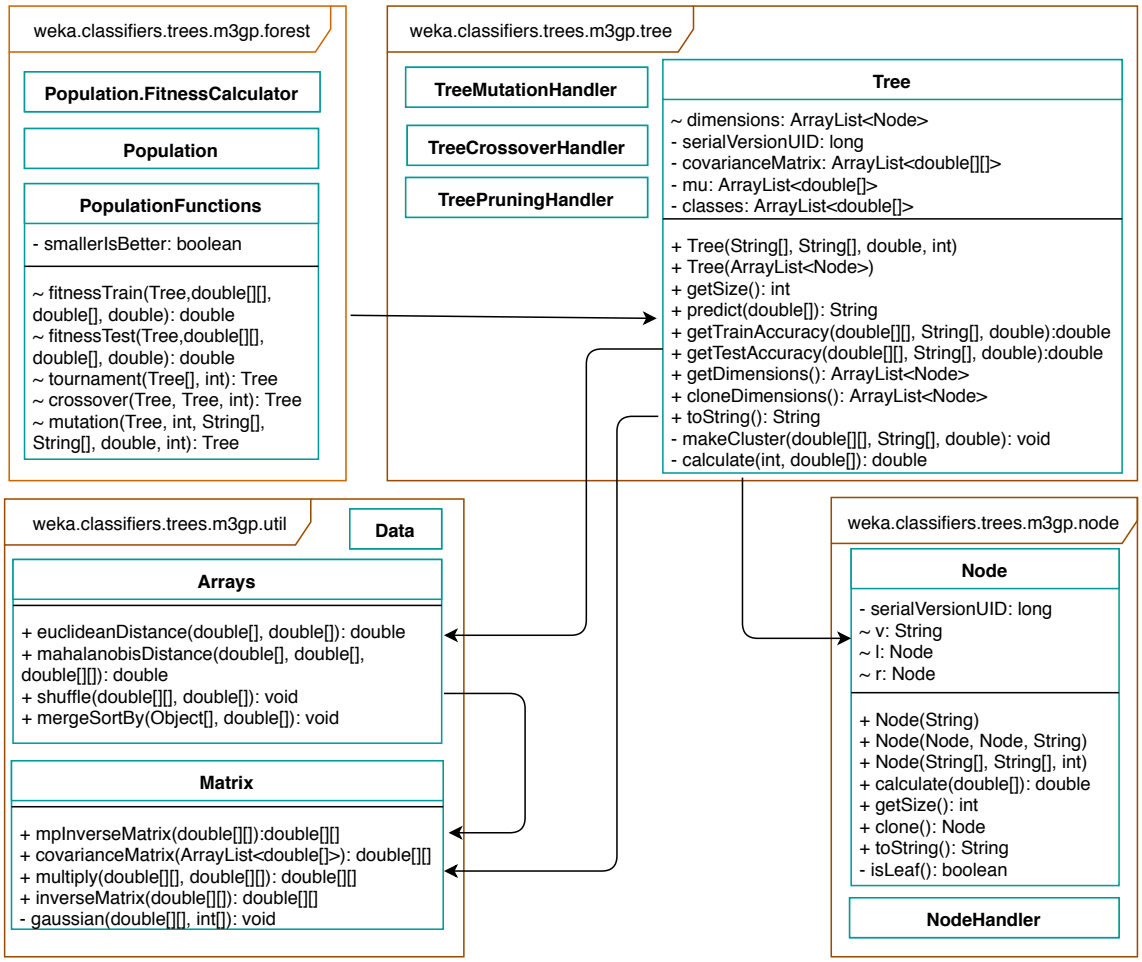

Figure A.2: Class Diagram section of the implementation used for the first stage of the project, referent to the evaluation of the population's individuals

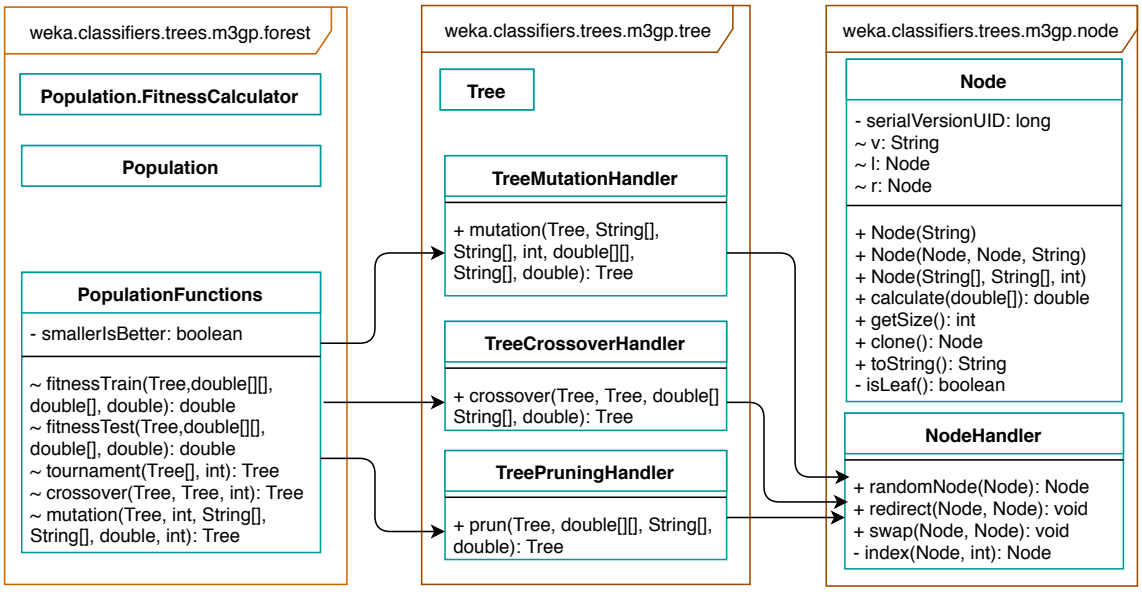

Figure A.3: Class Diagram section of the implementation used for the first stage of the project, referent to the usage of genetic operators

Appendix B

Results

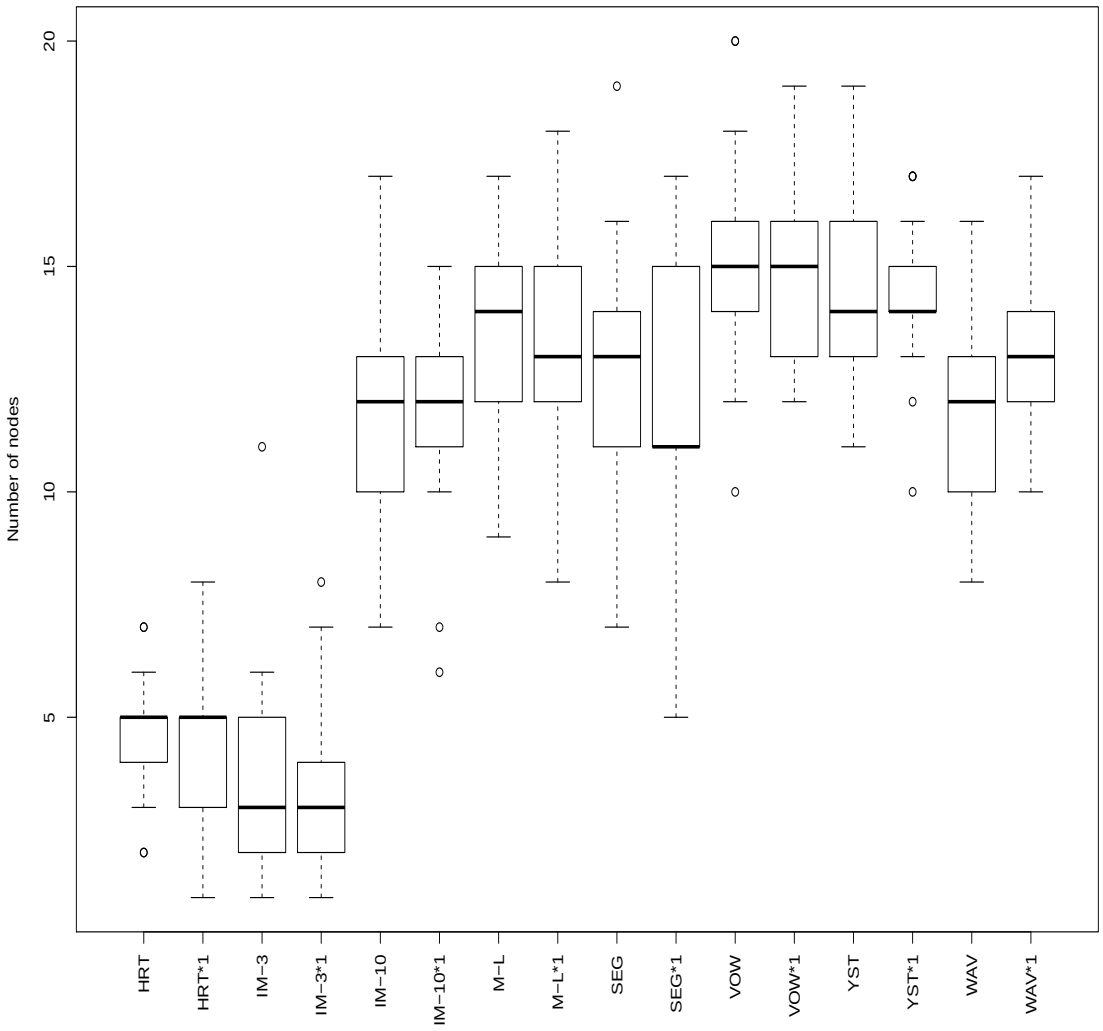

Figure B.1: Comparison of the number of dimensions between using the accuracy-based function (EA-FF) and using the distance-based function (ED1-FF), marked with *1, in the 8 datasets listed in 3.1.

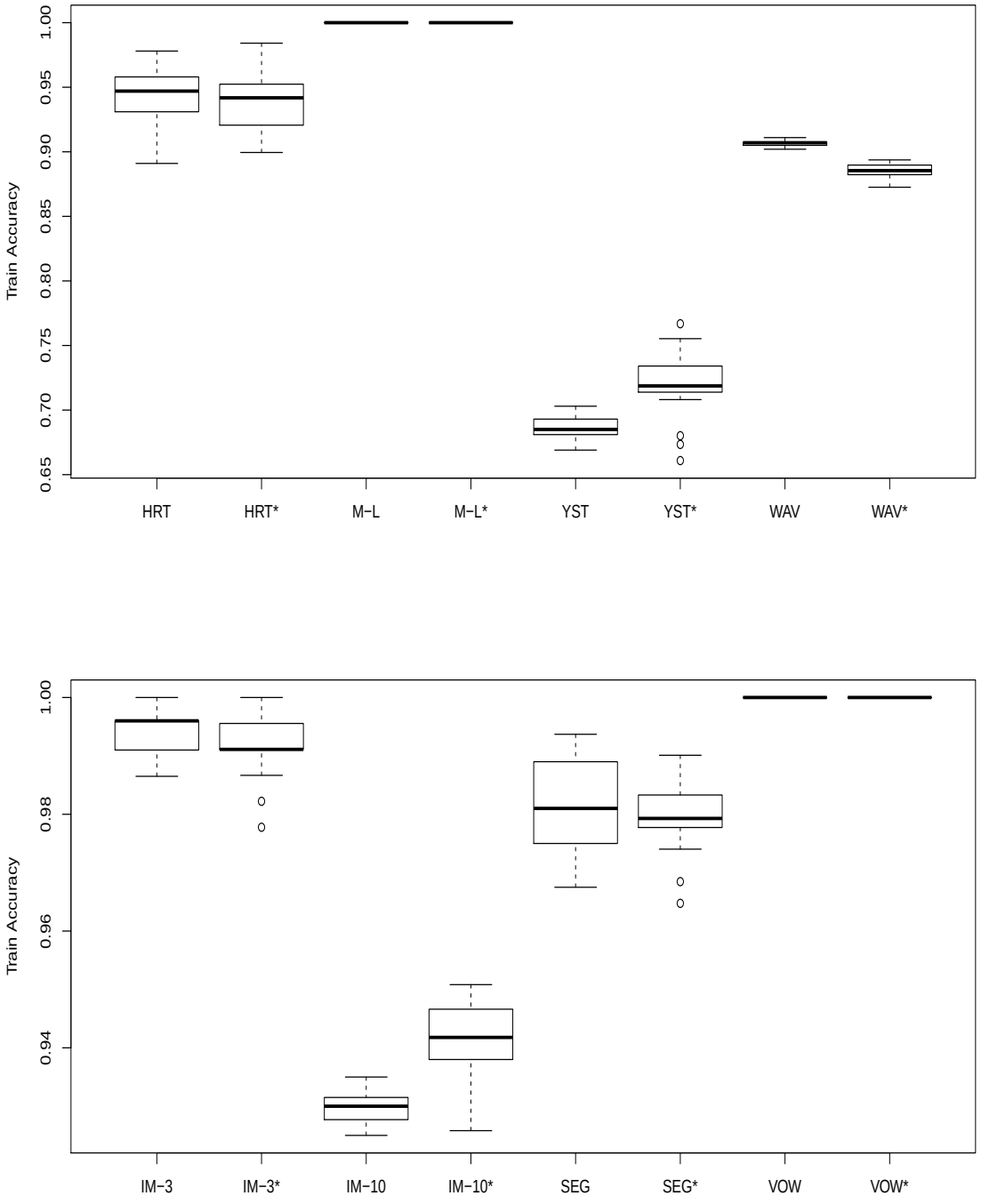

Figure B.2: Comparison of training accuracy between the original and our implementation of M3GP, marked with *, in the 8 datasets listed in 3.1.

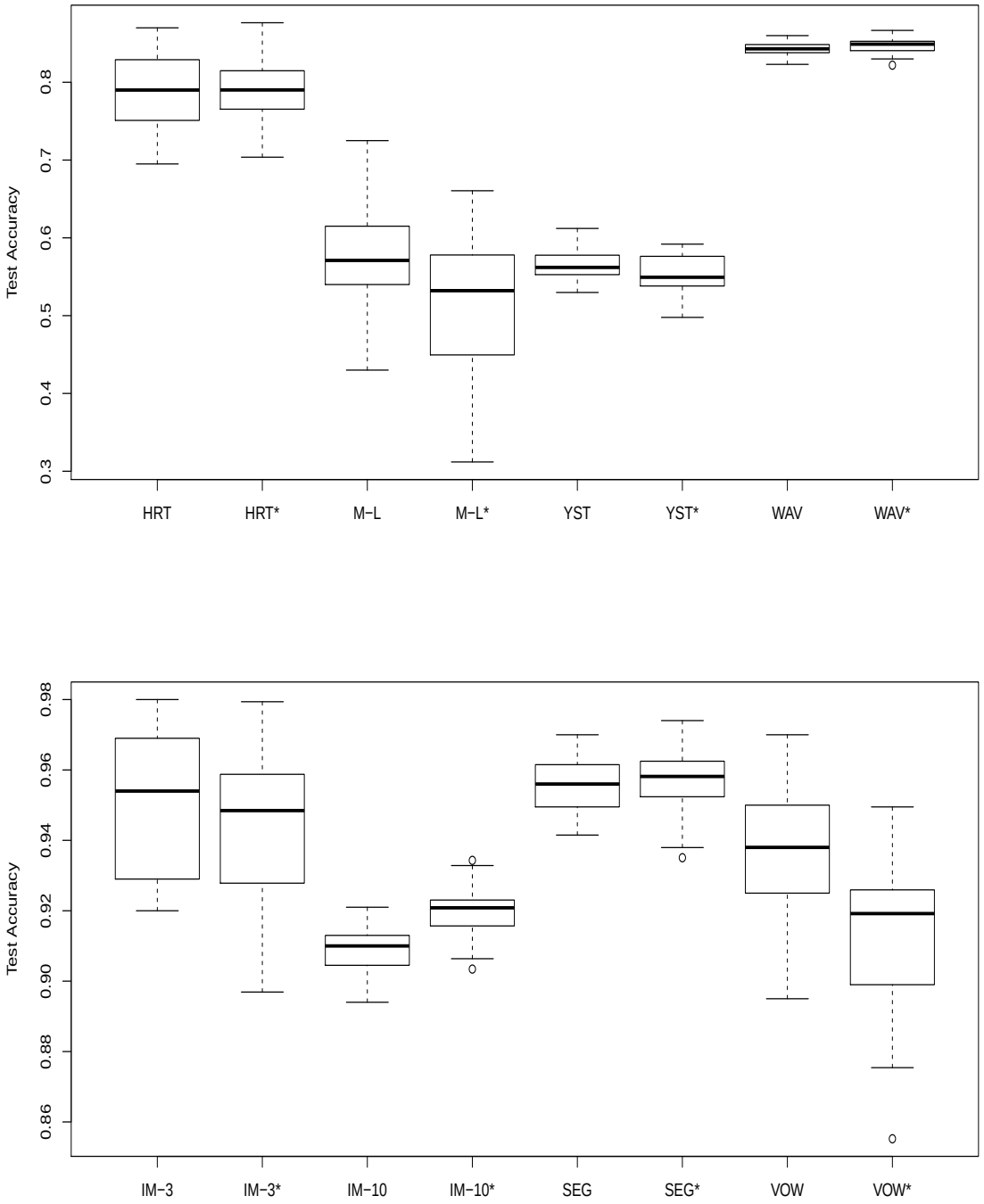

Figure B.3: Comparison of test accuracy between the original and our implementation of M3GP, marked with *, in the 8 datasets listed in 3.1.

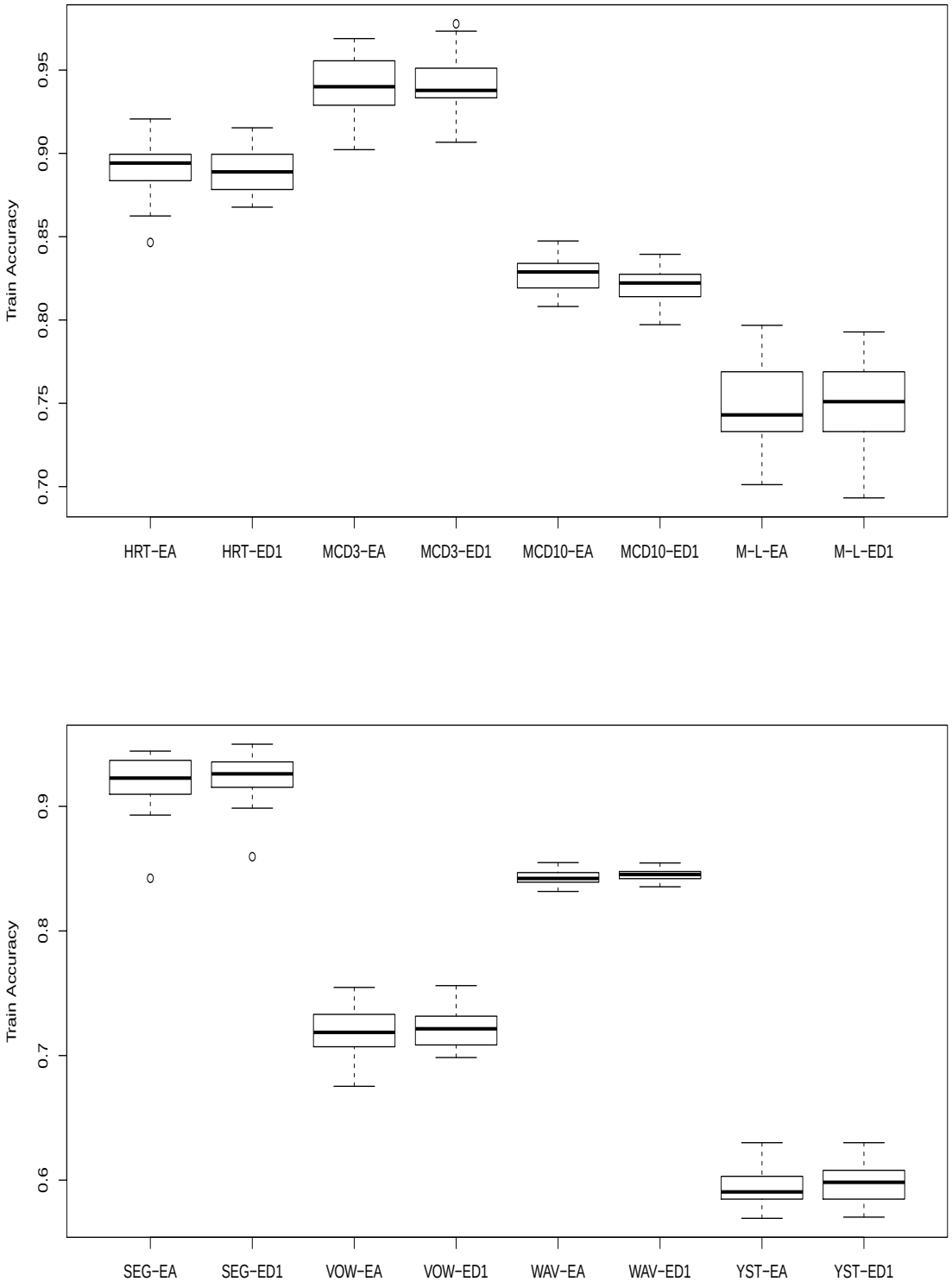

Figure B.4: Comparison of training accuracy on all datasets between using the Euclidean Accuracy fitness function (EA) and using the first distance-based fitness function (ED1)

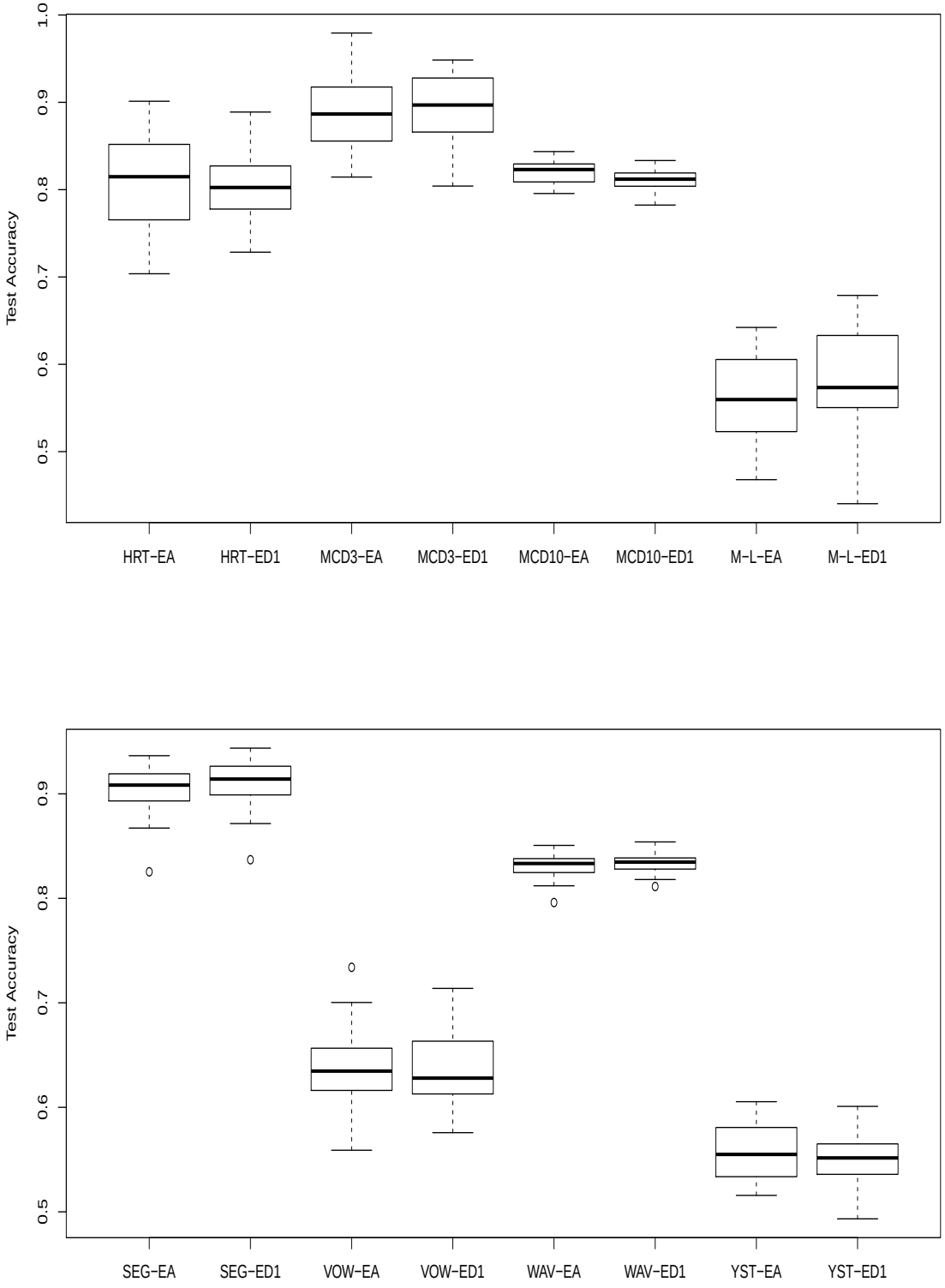

Figure B.5: Comparison of test accuracy on all datasets between using the Euclidean Accuracy fitness function (EA) and using the first distance-based fitness function (ED1)

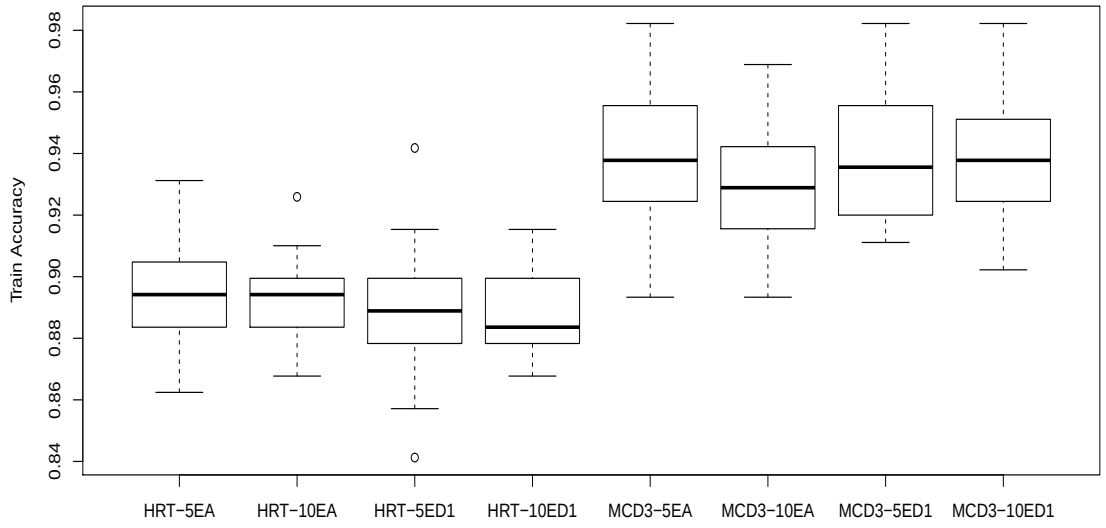

Figure B.6: Comparison of training accuracy on the HRT, and MCD3 datasets between using the Euclidean Accuracy fitness function (EA), and using the first distance-based (ED1) fitness function, while using five genetic operators (5) and ten genetic operators (10)

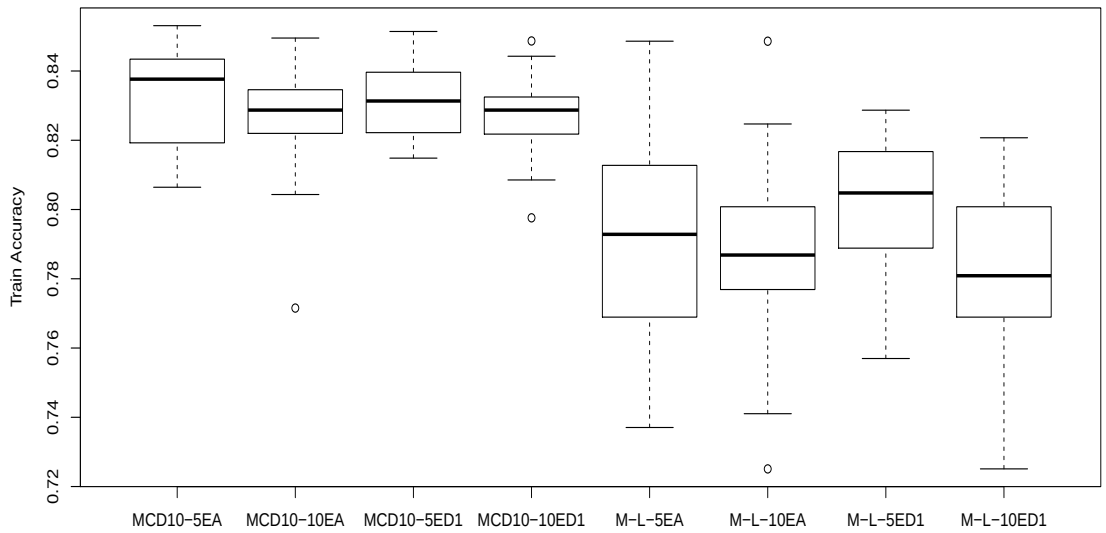

Figure B.7: Comparison of training accuracy on the MCD10, and M-L datasets between using the Euclidean Accuracy (EA) fitness function, and using the first distance-based (ED1) fitness function, while using five genetic operators (5) and ten genetic operators (10)

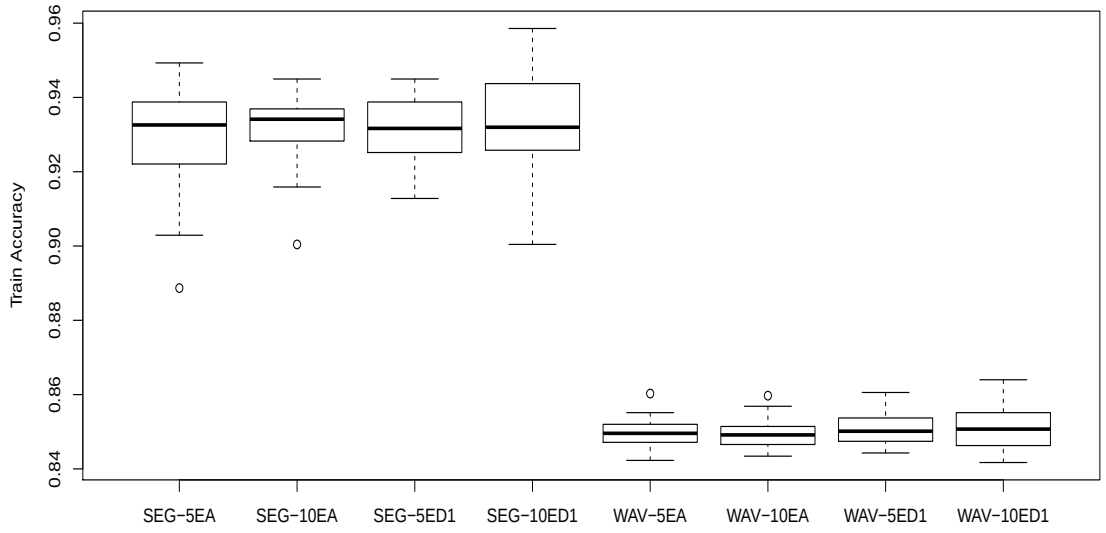

Figure B.8: Comparison of training accuracy on the SEG, and WAV datasets between using the Euclidean Accuracy (EA) fitness function, and using the first distance-based (ED1) fitness function, while using five genetic operators (5) and ten genetic operators (10)

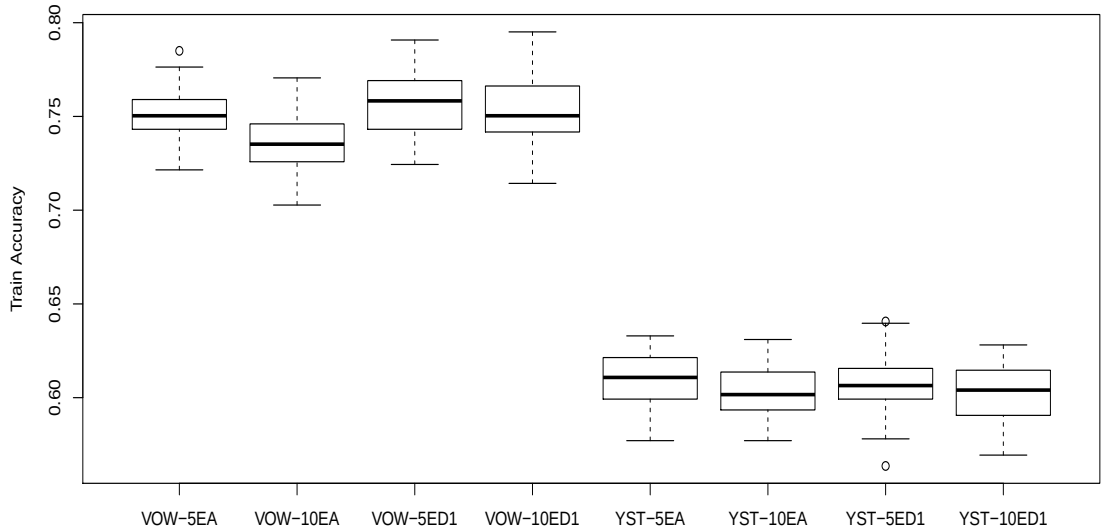

Figure B.9: Comparison of training accuracy on the VOW, and YST datasets between using the Euclidean Accuracy (EA) fitness function, and using the first distance-based (ED1) fitness function, while using five genetic operators (5) and ten genetic operators (10)

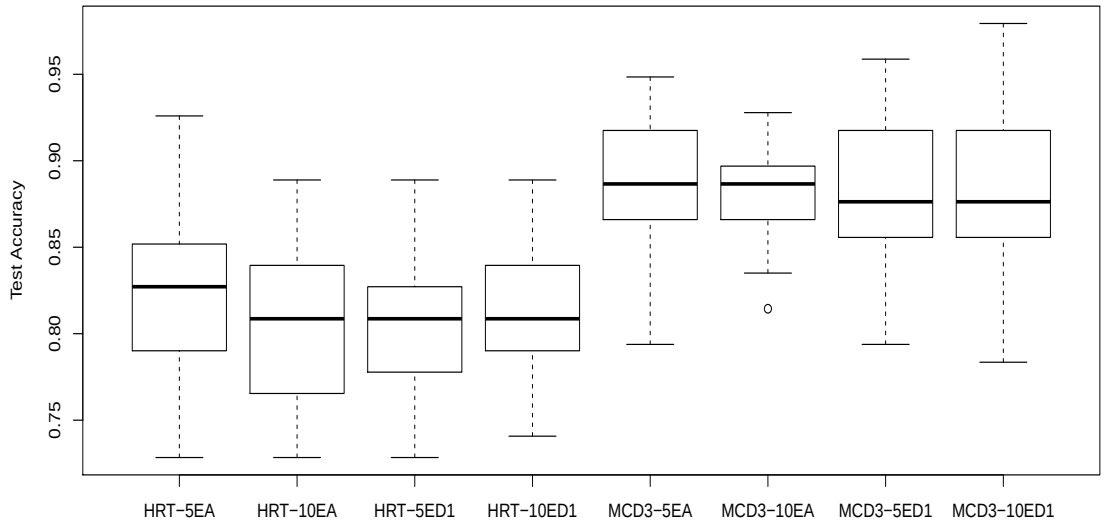

Figure B.10: Comparison of test accuracy on the HRT, and MCD3 datasets between using the Euclidean Accuracy (EA) fitness function, and using the first distance-based (ED1) fitness function, while using five genetic operators (5) and ten genetic operators (10)

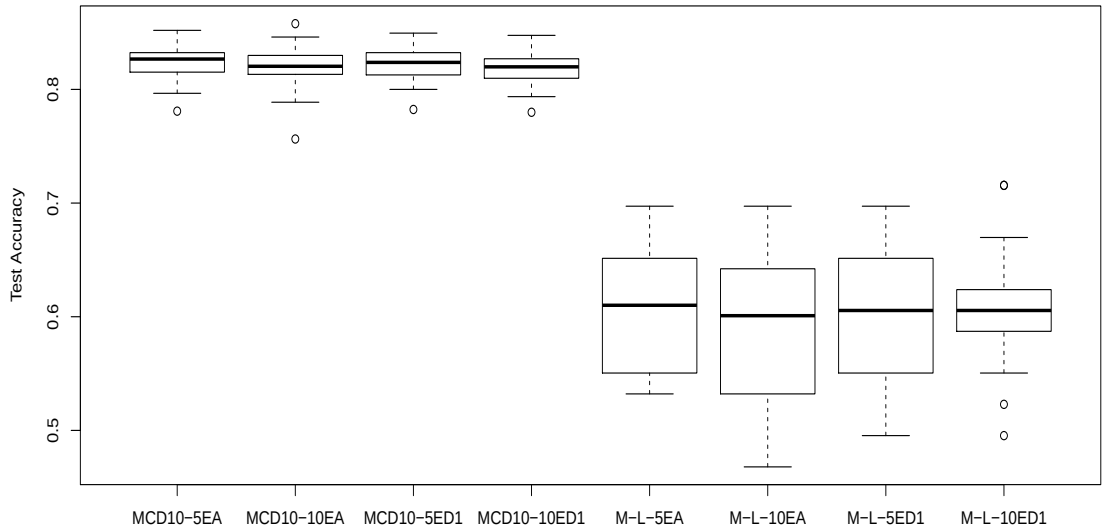

Figure B.11: Comparison of test accuracy on the MCD10, and M-L datasets between using the Euclidean Accuracy (EA) fitness function, and using the first distance-based (ED1) fitness function, while using five genetic operators (5) and ten genetic operators (10)

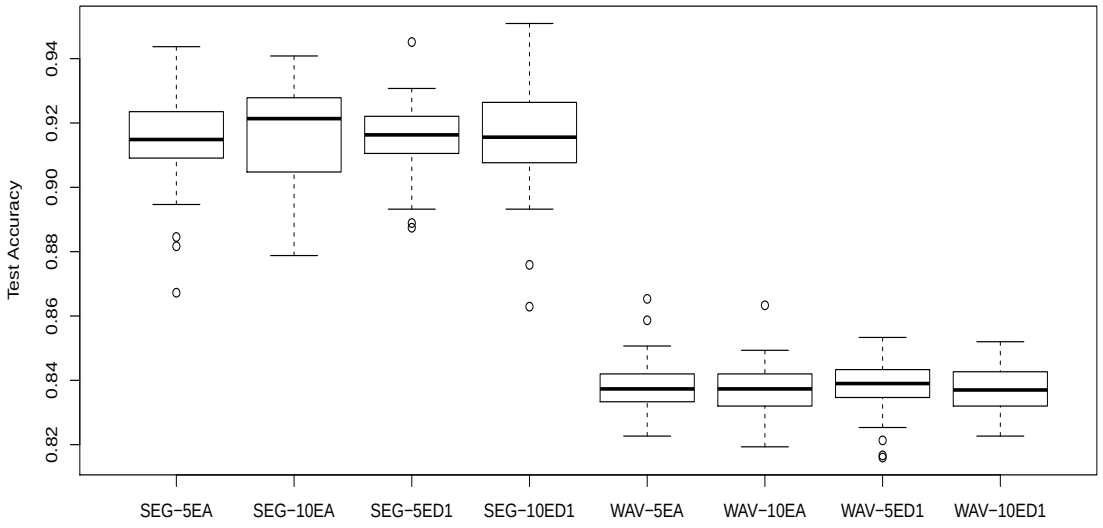

Figure B.12: Comparison of test accuracy on the SEG, and WAV datasets between using the Euclidean Accuracy (EA) fitness function, and using the first distance-based (ED1) fitness function, while using five genetic operators (5) and ten genetic operators (10)

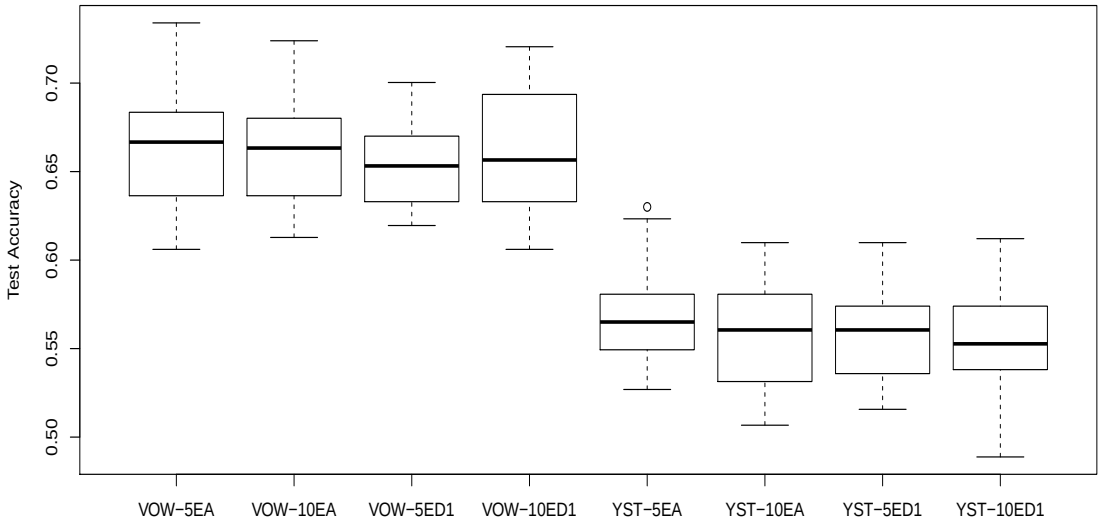

Figure B.13: Comparison of test accuracy on the VOW, and YST datasets between using the Euclidean Accuracy (EA) fitness function, and using the first distance-based (ED1) fitness function, while using five genetic operators (5) and ten genetic operators (10)